

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**FRAMEWORK FOR MANAGING METADATA SECURITY
TAGS AS THE BASIS FOR MAKING SECURITY DECISIONS**

by

Panagiotis Aposporis

December 2002

Thesis Advisor:
Second Reader:

Ted G. Lewis
Timothy E. Levin

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 2002	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Framework for Managing Metadata Security Tags as the Basis for Making Security Decisions.			5. FUNDING NUMBERS	
6. AUTHOR(S) Aposporis, Panagiotis				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) <p>This thesis presents an analysis of a capability to employ CAPCO (Controlled Access Program Coordination Office) compliant Metadata security tags as the basis for making security decisions. My research covers all the security aspects of the related technologies, such as XML, Web Services, Java API's for XML, .NET Architecture to help determine how security conscious enterprises such as the Intelligence Community can implement this approach in the real insecure world, with commercial off-the-self products, to meet their needs. There were many concerns about using the XML Metadata Label Tags as the basis for making security decisions, due to an un-trusted environment. By using appropriate trusted parts, when really necessary, and new technologies, we can find secure solutions for creating, storing and disseminating XML documents.</p> <p>Besides the theoretical research, this thesis also presents a prototype development of a Web Service that can handle most of the tasks (save, save locally, review etc), which are required to securely manage XML documents. In order to implement the above Web Service, open-source products, such as Java and Apache Tomcat Web Server, are used. These are not only available free, easily testable and commonly used, but they provide us with a great interoperability among almost all the platforms. The implementation can also be done by using other competitive technologies or platforms or can even use similar or related commercial products.</p>				
14. SUBJECT TERMS Metadata, Web Service, XML, XSL, DTD, Schema, SAX, Security Policy, XML Editor, XML Parser, Validate, Security Attributes, Labels, Objects.			15. NUMBER OF PAGES 288	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**FRAMEWORK FOR MANAGING METADATA SECURITY TAGS AS THE
BASIS FOR MAKING SECURITY DECISIONS**

Panagiotis Aposporis
Major, Hellenic Air Force
B.S., Hellenic Air Force Academy, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
December 2002**

Author: Aposporis Panagiotis

Approved by: Ted G. Lewis
Thesis Advisor

Timothy E. Levin
Second Reader

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis presents an analysis of a capability to employ CAPCO (Controlled Access Program Coordination Office) compliant Metadata security tags as the basis for making security decisions. My research covers all the security aspects of the related technologies, such as XML, Web Services, Java API's for XML, .NET Architecture to help determine how security conscious enterprises such as the Intelligence Community can implement this approach in the real insecure world, with commercial off-the-self products, to meet their needs. There were many concerns about using the XML Metadata Label Tags as the basis for making security decisions, due to an un-trusted environment. By using appropriate trusted parts, when really necessary, and new technologies, we can find secure solutions for creating, storing and disseminating XML documents.

Besides the theoretical research, this thesis also presents a prototype development of a Web Service that can handle most of the tasks (save, save locally, review etc), which are required to securely manage XML documents. In order to implement the above Web Service, open-source products, such as Java and Apache Tomcat Web Server, are used. These are not only available free, easily testable and commonly used, but they provide us with a great interoperability among almost all the platforms. The implementation can also be done by using other competitive technologies or platforms or can even use similar or related commercial products.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	DEFINITION OF THE PROBLEM.....	1
A.	INTRODUCTION.....	1
	1. Background.....	1
B.	BRIEF OVERVIEW OF THE SAIC – IC XML WORKGROUP FINAL REPORT	2
	1. Content Manipulation	2
	2. Content Storage	3
	3. Content Delivery.....	3
	4. Conclusions	4
C.	DESCRIPTION OF THE PROBLEM AND A PROPOSED APPROACH.....	4
D.	OUTLINE	9
II.	CURRENT STATE OF SECURITY IN XML - METADATA	11
A.	DEFINITION OF THE TERMS	11
	1. XML, XSL and SCHEMA	11
	2. Metadata.....	12
	3. SOAP and Web Services	14
	4. CORBA.....	15
B.	SURVEY OF SECURITY ISSUES IN RELATED TEC HNOLOGIES AND RESEARCH LITERATURE SURVEY	16
	1. WS-Security	16
	2. Microsoft’s .NET Framework Security	18
	a. <i>Introduction</i>	18
	b. <i>Code Access Security</i>	19
	c. <i>Role-based Security</i>	20
	3. Java Security	21
	a. <i>Introduction</i>	21
	b. <i>Basic Concepts</i>	22
	c. <i>New Enhancements of the Java 2 SDK</i>	23
	4. COM & DCOM.....	24
	a. <i>Introduction</i>	24
	b. <i>Security</i>	25
C.	IC METADATA STANDARD FOR PUBLICATIONS	26
III.	DBMS ARCHITECTURES.....	29
A.	OVERVIEW OF THE BASIC SECURE DBMS	29
	1. Historical Background.....	29
	2. Woods Hole Architectures	30
	3. Trusted Subject Architectures.....	32
B.	ANALYSIS AND REQUIREMENTS OF DBMS ARCHITECTURES ..	33
	1. Basic Requirements.....	33

2.	XML and DBMS Architectures	35
C.	RESEARCH ON RELATED DBMS PRODUCTS	38
1.	Oracle 9i.....	38
2.	IBM DB2 Universal Database V7	40
3.	Sybase Adaptive Server Enterprise (ASE) 12.5	42
D.	THREE-TIER ARCHITECTURE.....	44
IV.	IMPLEMENTATION.....	47
A.	SECURITY ARCHITECTURE OF THE PROTOTYPE DEVELOPMENT	47
1.	Introduction	47
2.	Security Policies	49
B.	STRUCTURE OF THE PROTOTYPE DEVELOPMENT.....	50
1.	Main Idea.....	50
2.	Description of System's Logic—Flow Diagrams.....	50
3.	Analysis of the Java Servlets	58
a.	<i>Login.class</i>	58
b.	<i>UserOptions.class</i>	58
c.	<i>UserSelection.class</i>	59
d.	<i>ImportToSave.class</i>	59
e.	<i>SaveFile.class</i>	59
f.	<i>UpdateDb.class</i>	61
g.	<i>OpenFile.class</i>	61
h.	<i>ImportFile.class</i>	62
i.	<i>FindUrl.class</i>	62
j.	<i>Logout.class</i>	63
V.	EXPERIMENTATION.....	65
A.	DEMONSTRATION SCENARIOS	65
1.	Introduction	65
2.	Scenario 1 – Internal User Stores a Document to the System	65
3.	Scenario 2 – Internal User Retrieves a Document from the System	68
4.	Scenario 3 – External User Retrieves a Document from the System	71
B.	SUBJECTIVE PERFORMANCE EVALUATION.....	74
C.	CONCLUSIONS - LESSONS LEARNED	75
APPENDIX A.	APPLICATION PROGRAMMING INTERFACE (API)	77
	Package dbsection	78
	Hierarchy for Package dbsection	80
APPENDIX B.	JAVA CODE	175
LIST OF REFERENCES	269
INITIAL DISTRIBUTION LIST	273

LIST OF FIGURES

Figure 1.	Basic Proposed Architecture.....	6
Figure 2.	Server Side Proposed Architecture.....	7
Figure 3.	Client Side.....	8
Figure 4.	WS Security Summary.....	17
Figure 5.	Kernelized DBMS	30
Figure 6.	Distributed DBMS	31
Figure 7.	Cryptographic Integrity Lock DBMS	32
Figure 8.	Trusted Subject Architectures	33
Figure 9.	Security Architecture.....	47
Figure 10.	Login Flow Diagram	51
Figure 11.	Main Menu Flow Diagram	52
Figure 12.	Save to Database Flow Diagram.....	54
Figure 13.	Open File Flow Diagram.....	55
Figure 14.	Open from External Database Flow Diagram	56
Figure 15.	Login Screen.....	65
Figure 16.	First Option in the Main Menu	66
Figure 17.	Browse Local File System.....	67
Figure 18.	Uploading and Parsing File Process	67
Figure 19.	Second Option in the Main Menu	68
Figure 20.	Available Files to Open.....	69
Figure 21.	Choose an Editor Screen	69
Figure 22.	XMLMind Editor Opened the File	70
Figure 23.	Applet Based Editor Opened the File	71
Figure 24.	Third Option in the Main Menu.....	72
Figure 25.	Choose an Associated Server Screen	72
Figure 26.	Available Files on an Associated Server	73

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Dr. Cynthia Irvine for introducing me to the field of computer security and for her professional guidance and direction during the entire length of my studies at the Naval Postgraduate School. I would also like to thank my Thesis advisors Ted Lewis and Tim Levin for their guidance, support and patience. Their experience and expert knowledge inspired me to reach beyond my previous limits and capabilities. I am also grateful to Ron Russell, my English editor, for his support, patient and excellent corrections. In addition, I sincerely thank the Hellenic Air Force General Staff for sponsoring me for this course and the entire faculty and staff at the Naval Postgraduate School for helping me successfully complete the curriculum.

Last but certainly not least, I am indebted to my loving family, my lovely wife Haroula, and my wonderful children, John and Dina, who provided me with unlimited support and love during this research. Without their support, none of my accomplishments would have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. DEFINITION OF THE PROBLEM

A. INTRODUCTION

1. Background

The eXtensible Markup Language (XML) is a text and data-formatting language that has a tag-based syntax very similar to the Hyper Text Markup Language (HTML) syntax, but much more capable and much more flexible. It not only prescribes text styles but also defines data types for cross platform communication. XML documents contain only data, so applications that process XML documents must decide how to display the document's data. For example a Personal Digital Assistant (PDA) may render an XML document differently than a wireless phone or a desktop computer would render that document.

XML permits document authors to create "markup" for virtually any type of information. Markup is notation that provides information to an XML parser on how to parse, or read, an XML document; which parts of a document to skip; and which parts of a document to hand off to another application. Consequently, this extensibility enables document authors to create entirely new markup languages for describing specific types of data, including mathematical formulas, chemical molecular structures, music etc. Based on that powerful capability, the Intelligence Community (IC) recently began studying XML and its potential use as the main technology to create, manage and disseminate intelligence's content.

At the end of 1999 a Final Report [24] was prepared under the direction of the Office of Advanced Analytical Tools (AAT) (Department of the Central Intelligence Agency), by the Science Applications International Corporation Applied Content Technologies Team (SAIC) for the IC XML Study Group. The purpose of that report was

to present the findings of the development of a prototype with emphasis on the technical highlights, advantages, disadvantages and shortfalls of utilizing XML for production, storage and dissemination of intelligence data. Additionally it provides

recommendations for utilizing XML technology in the Intelligence Community.

The findings of that report as concerns the applicability of the XML to the Intelligence's Community documents were encouraging and highly promising for the future. More specifically, the XML was found well suited for creating, managing and disseminating the intelligence content. XML provides several solutions to the form of authoring, manages highly granular level data from many different sources, and finally enables the dissemination of user specific views and of data based on target security domain. Even though this report was published about three years ago and much has changed in the related technologies, it contains many concepts worthy of mentioning and analyzing.

B. BRIEF OVERVIEW OF THE SAIC – IC XML WORKGROUP FINAL REPORT

1. Content Manipulation

The proposed Content Manipulation in the IC XML Workgroup Final Report [24] has two distinct sub-functions: the authoring environment, and the comment/review process. In the authoring environment, the analyst can use an XML editor to create an XML document based on the rules in a Document Type Definition (DTD). This DTD provides the structure and the business rules that must be followed as the XML content is created. The interactive parser is responsible for validating the XML instances created in this environment and to provide a valid XML document. There is also a potential use of a non-XML editor, which can create “structured” documents, using formatting styles from templates that are generated from the DTD. Due to the large number of powerful, and in most cases, free XML editors that have been developed, non-XML editors will not be analyzed further in this document. The main feature that the XML Editor can provide is that it can interface with the dissemination/delivery environment by serving as an on-call editor that can be called directly from a web browser.

In the second function, which was called the “comment/review process,” a potential reviewer outside the authoring environment can access, review and comment on the content of an XML document. It was also suggested that the process be done via a web browser, not only the whole document but its fragments, and the comments of the

reviewer be stored as custom attributes and maintained in the content repository, separate from the original document, which allows many reviewers to provide comments without altering the original document.

2. Content Storage

The content storage mechanism is separated into three functional areas: a) the content management system, b) a content repository and c) all other internal and external sources. The content management system must enable general document management capabilities, such as security, access control, check in/out and version control history. In addition, this system will provide the interface through which the XML editor accesses the repository, and the users manage and reuse the document components. The system also controls the access to documents in the repository for services such as comment/review and modifying documents.

The main component of the content storage is an object-oriented database, which is controlled by the management system. An object-oriented database is preferred as it is better suited for hierarchical structures, such as those typically produced by XML. This content repository is used to house the XML content at a very granular level while allowing for check-in, checkout and versioning of the content. It is also very advantageous for that object-oriented database to be able to connect to other data sources, especially relational databases.

3. Content Delivery

The content delivery functional area consists of three basic parts:

- domain filtering,
- internal user functions, and
- external user functions.

The production management, staff and the authors or analysts are the internal users who create and distribute the intelligence documents stored in the content repository. Anyone else who is outside the production environment and who does not contribute directly to the production of the distributed content is the external user. External user functions include search and retrieval of distributed content and external repositories and the capability to capture distributed content locally.

The main part of the content delivery is the domain filter, which consists of scripting filters designed to separate XML instances into various versions based on security classification metadata. When an XML document is exported from the database, the data is processed through the appropriate filter and distributed accordingly.

For both internal and external users, the necessary functions are provided through a Web browser interface of various pages that are based on user profiles and security access privileges. When the IC XML Study Group published their report the XML was a new work in progress and so the technologies to develop, distribute and browse XML contents was limited. Consequently, the authors were forced to transform the XML documents into HTML using XSL style sheets and to send them as styled HTML to the end user.

4. Conclusions

Clearly, security will always be a major factor in the IC, currently and in future environments. The IC XML Study Group report correctly concluded that nothing in XML inherently prohibits its use in the current security environment. Moreover, it also has a great capability to attach security attributes to granular elements. This capability is much more powerful than those in HTML documents. In addition, if the security markings are stored as attributes rather than actual content, then changing markings guides is only a matter of changing style sheets rather than content. Security markings are a great example of why XML is better than HTML for storage of intelligence content.

Generally speaking, XML shows great applicability by providing a vendor-neutral, non-proprietary format for exchanging content, which enables better communications among the members of the IC and strives for better interoperability with its customers.

C. DESCRIPTION OF THE PROBLEM AND A PROPOSED APPROACH

Among the main goals of the prototype that the IC XML Study Group's report was trying to achieve was to suggest a combination of

the latest and best commercial-off-the-self XML technology and
explore and develop this technology into a system that allows users

to author, store and disseminate disparate types of data in XML format.

Some of the “latest and best” products of that period were examined and some testing was done mainly trying to prove that all could work together and could provide the desired results. Even though each of those products was really among the best of all the others within the same functional family, many problems and difficulties existed when they had to cooperate with each other. Furthermore, the XML technology was not very mature and fully developed at that period. This made their job much more difficult because everything had to be converted to HTML in order to be processed and delivered.

Besides the above functional and operational factors, another major problem was the lack of security of the whole process because most of the elements used or proposed were commercial non-open source products. In addition, there were security concerns regarding the XML documents themselves during their storage or dissemination as well as when the system must decide if a document is releasable to a user and to what extent (the entire document or just paragraphs). There is of course a general reference for those “domain filters” that must “separate XML instances into various versions based on security classification metadata.” There is also a general reference for these functions (comment/review, print save locally) that must be provided through a Web browser interface, but the report also states “the current status of XML as a work in progress, the software support to develop, distribute and browse XML content is limited.”

Recently much has changed, so the new technologies can work in a distributed environment with the maximum interoperability. One of the major changes was the XML technology, which became more mature and every day more and more vendors and developers chose it as the base of their applications. Moreover, the XML is integrated and is improved in a new technology called XML Web Services. The real power of the XML Web Services is that it lets applications share data and invoke capabilities from other applications without regard to how those applications were built. XML is also independent and can work with any operating system or platform they run on and any devices are used to access them. While XML Web Services remain independent of each

other, they can loosely link themselves into a collaborating group that performs a particular task.

The approach proposed in this thesis is to use a Web Service as the manager, which uses the enormous capabilities of the XML in conjunction with its Metadata Labels as the base to make the required security decisions. In that way, we obtain all the advantages of the XML and the Web Services as well as maintaining the existing systems or applications due to the fact that both XML and Web Services are completely independent of the other parts of the entire system. There is a serious security concern for those Metadata Label Tags and how they can be trusted in the real world's insecure Internet environment. By using some of the widely used secure solutions (like SSL, TCB, DBMS), we can build a fully functional, interoperable and virtually secure system for use by any large-scale enterprise. The proposed architecture is shown in Figure 1.

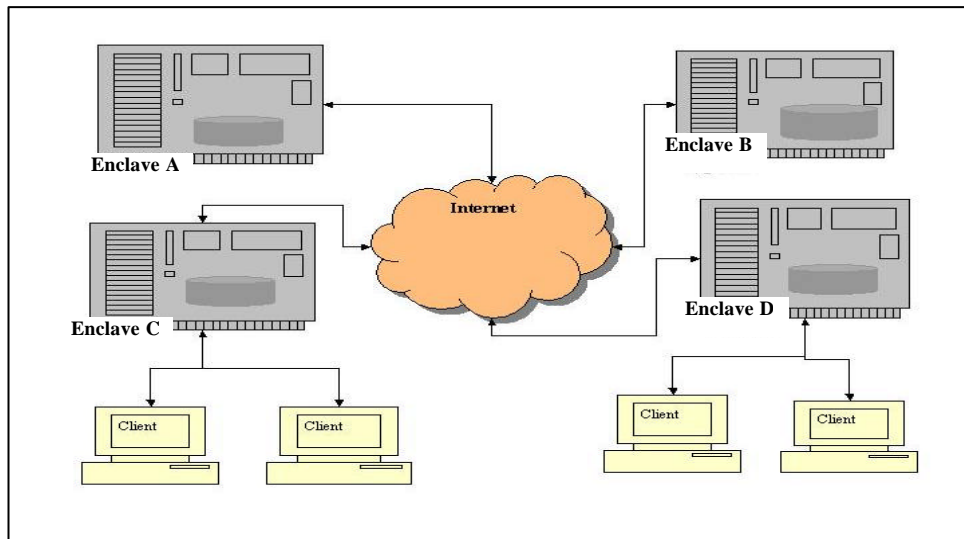


Figure 1. Basic Proposed Architecture

The main idea, as shown in Figure 1, consists of a number of enclaves, which are interconnected through the Internet using any of the common protocols, such as Hyper Text Transfer Protocol (HTTP) or its secure version (HTTPS). Over the HTTP or HTTPS a relatively new protocol, called Simple Object Access Protocol (SOAP) is running, which can be roughly described as a combination of HTTP, XML and the Remote Procedure Calls (RPC). Any of the usual technologies that have been used so far,

for authorization and authentication, between the enclaves can be used without affecting the main logic or operation of the system.

In every enclave, the XML Web Service is running on a dedicated Web Server, which of course can be any of the common commercial ones or a modified open-source server. The main tasks of this Web Service can be divided into three main categories. The first one is the Authentication and Authorization of a potential client, which can also be a single user or another Web Service, running on a Web Server in one of the interconnected enclaves. The second category consists of the tasks that the system must perform when the client wants to work on an XML document that is stored in its local database. That means a direct communication with the local Database Management System (DBMS) must be established so the desired documents are accessible. The last category includes all the tasks that must be performed when the client wants to access an XML document that is stored outside of its own database. In that category, a connection with the appropriate Web Service in the specific server of the enclave must be established, using the protocols mentioned before.

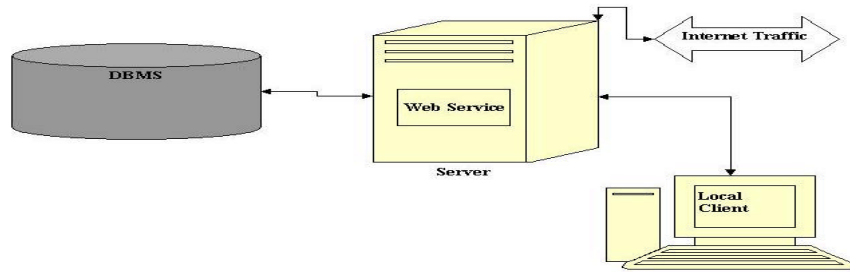


Figure 2. Server Side Proposed Architecture

The most common solution to interface with the Web Service is by using any browser. Since most people are familiar with browsers nowadays, no special training is required. Interaction between the user and the system is done through appropriate pages that are generated dynamically by the Web Service according to the user's choices and the enforced policy. Since no special features from the browser are needed, a commercial off-the-self product can be used or a "hand-made" open source instance can also be used.

When the user, via a Web browser, accesses an XML document, an XML Editor must be used to handle the document itself. This system has enormous flexibility and interoperability because the user's choice of editor has no restrictions and can be either a commercial off-the-self product or an open source. For content creating tasks, the editor may also parse and check each document to determine if it is valid according to the Intelligence's Community XML DTD (Document Type Definition) for Security Markings. The same checking is also done on the server side before data is accepted. Recently that DTD has under gone a complete change and has been renamed "DED (Data Element Dictionary) for the IC security attributes." An overview of that DED can be found in the following chapters of this paper.

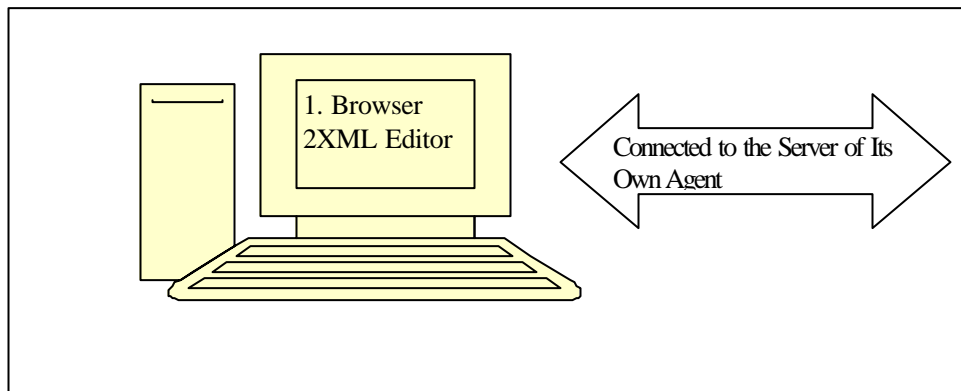


Figure 3. Client Side

A Web Service can be written in any of the common languages (e.g. Java, C++, Visual Basic) by utilizing any of the new technologies, such as the Microsoft's ". NET" framework, or the Sun's Java WSDP (Web Service Development Package). When there is a need for a high assurance system, which also has to be expandable and interoperable, using technologies that are open source is preferable so that they can be tested and their security can be verified either by using formal or non-formal methods. For that reason, Sun's Java WSDP has an advantage over other technologies and has already been used by other major vendors (IBM, Oracle) as the base for their applications or development tools.

In my implementation, Sun's Java WSDP is used as the framework to implement the Web Service because Java APIs for XML allow us to write our applications entirely

in the Java programming language, thus taking advantage of the large number of the features this language provides. Besides being an open source free technology, the most important features of the Java APIs for XML are that they all support industry standards, thus ensuring interoperability. Various network interoperability standards groups, such as the World Wide Web Consortium (W3C) and the Organization for the Advancement of Structured Information Standards (OASIS) have defined standards so that businesses that follow these standards can make their data and applications complementary.

Another feature of the Java APIs for XML is that they allow a great deal of flexibility. Users have flexibility in how they use the APIs. For example, JAXP (Java API for XML Processing) code can use various tools for processing an XML document, and JAXM (Java API for XML Messaging) code can use various messaging protocols on top of SOAP. Implementers have flexibility as well. The Java APIs for XML define strict compatibility requirements to ensure that all implementations deliver the standard functionality, but they also give developers a great deal of freedom to provide implementations tailored to specific uses like those for the Intelligence Community.

D. OUTLINE

The remainder of this paper is organized as follows. Chapter II. “Current State of Security in XML – Metadata” describes the terms and the current state of the security in many of the new technologies as well as the main points of the IC Metadata Standards for Publications. Chapter III “Metadata DBMS Architectures” refers to the basic secure architectures, to a proposed architecture appropriate for metadata, and to the concepts that have already been implemented by some of the most specialized vendors. In Chapter IV “Implementation (Prototype Development)” a review of the 3-tier architecture is made, as well as a complete analysis of the Java classes being used, along with their relations. Chapter V, “Experimentation,” concerns the testing of the system with different cases and also describes the lessons learned and future work that could be done. Finally Appendix I, completely documents the classes used as the official “javadoc” tool of the Java programming Language produces and Appendix II presents a complete listing of the code.

II. CURRENT STATE OF SECURITY IN XML - METADATA

A. DEFINITION OF THE TERMS

1. XML, XSL and SCHEMA

XML is a subset of the Standard Generalized Markup Language (SGML) defined in ISO standard 8879:1986 that is designed to make it easy to interchange structured documents over the Internet. XML files always clearly mark the beginning and end of each of the elements of an interchanged document. XML restricts the use of the SGML constructs to ensure that fallback options are available when access to certain components of the document are not currently possible over the Internet. With XML, we can use any tags we want, and the browser does not automatically understand the meaning of these tags. For example the tag <body> could mean an HTML text body or perhaps the human body in a medical article. Because of the nature of XML, no standard way to display an XML document exists.

In order to display XML documents, having a mechanism to describe how the document should be displayed is necessary. One of these mechanisms is Cascading Style Sheets (CSS), but the eXtensible Stylesheet Language (XSL) is the preferred style sheet language of XML. In addition XSL is far more sophisticated than the CSS used by HTML. XSL consists of two parts, a method for transforming XML documents and a method for formatting XML documents. As an example, we can think of XSL as a language that can transform XML into HTML, a language that can filter and sort XML data, and a language that can format XML data, based on the data value, like displaying negative numbers in red. XSL can be used to define how an XML file should be displayed by transforming the XML file into a format that is recognizable to a browser. Normally XSL does this by transforming each XML element into an HTML element because HTML is a browser recognizable format. XSL can also add completely new elements into the output file, or can remove elements. It can rearrange and sort the elements and test and make decisions about which elements to display, and a lot more.

An XML document optionally can reference another document that defines that XML document's structure. This referenced document is either a Document Type

Definition (DTD) or a Schema. A DTD expresses the set of rules for the document structure using an Extended Backus-Naur Form (EBNF) grammar. Unlike DTDs, Schemas do not use EBNF grammar. Instead they use XML syntax and are actually XML documents that programs can manipulate like other XML documents. Schemas are more flexible and more powerful than DTDs, and many researchers in the XML community believe that Schemas will prevail over DTDs. When an XML document references a DTD or Schema, some parsers, which are also called validating parsers, can read the DTD or the Schema and check the XML document according to the structure that the DTD or Schema defines. If the XML document conforms to the DTD/Schema, then the XML document is “valid.” Those parsers that cannot check for document conformity against DTD/Schemas are called non-validating parsers. If an XML document can be processed successfully by a validating or a non-validating parser, which means that the document is syntactically correct, then that document is called “well formed.” By definition, a valid XML document is also well formed.

2. Metadata

A common short definition of metadata is that it is “data about data.” Actually, metadata can be data about almost anything. What makes it metadata is its purpose and usage rather than its content or structure. Most often, metadata is designed to support people or programs in locating and retrieving information resources. A piece of data can be metadata to one application, and just data to another. Metadata is relatively short, has a simple structure, and is so familiar that one may not realize that he or she uses metadata every day. The most common example is when using a collection of tapes or CDs. If all of them were in blank boxes, and one had to play each of them to see what was recorded on it, then finding a particular film or track would be a long and difficult job. However, if one places a label with the name of the tape or the CD and a list of the songs on each of them, then picking out the right one immediately becomes very much easier. Extending this example to a large library or the World Wide Web, the problem of finding exactly what you are looking for is clearly enormous, but the basic concept of the solution remains the same. Make some simple, relevant, searchable information available, together with location information, and searching and retrieval becomes much easier.

Unfortunately, when the information pool is very large, a large number of relevant results can be found, which must be filtered and ranked according to the specific user's needs.

XML metadata can take many different forms. It may be embedded in an XML document alongside the information it is about, or it may be held in a separate XML document. In the second case the XML metadata could identify which information it is about by using, for example, a Unified Resource Identifier (URI). In order to understand better the breadth of metadata some examples of different kinds are presented below:

- *Annotation:* These are side notes added to a document for a specific purpose, and they will be read by some or by all of the readers but at different times and for different purposes.
- *Cataloguing and Identification:* In this kind of metadata there is an association between specific properties and their values with whatever the metadata is about. For example, a music store catalogue record for a music CD, gives its title, singer and publisher. In XML applications, this metadata is usually information about the properties of information resources, leading to the general name “resource based metadata.”
- *Subject Indexes:* This refers to metadata that represents subjects and their interrelationships, and also usually designates specific information resources as belonging to these subjects. A closely related kind of metadata is the already mentioned example with the tapes and CDs. Those kinds of metadata are usually referred to as “subject based metadata.”
- *Cross-references:* These are an important kind of metadata for long-lived collections of documents where documents are often cross-referenced in ways their original authors did not foresee. In complex independent collections, such as legal codes and cases, cross-references become structural mapping.

In practice, most technologies include aspects of both resource based and subject based metadata. This is partly because if you want to represent a subject inside a computer, you will probably end up doing so either by identifying the subject with an

information resource such as a thesaurus entry or by using a name for the subject as a value of a property. In the music store example, the catalogue record can also give extra information for each song contained in it. Moreover, if you have many resources with properties, you will want to control and structure which properties are used, how they relate to each other, and which of them will take on a life of their own.

3. SOAP and Web Services

SOAP stands for the Simple Object Access Protocol. SOAP is based on XML and describes a messaging format for machine-to-machine communication. SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define any application semantics, such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to Remote Procedures Call (RPC). SOAP consists of three parts: a) the SOAP envelope construct defines an overall framework for expressing what is in a message, who should deal with it, and whether it is optional or mandatory, b) the SOAP encoding rules define a serialization mechanism that can be used to exchange instances of application-defined data types, and c) the SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses. Although these parts are described together as part of SOAP, they are functionally orthogonal. In particular, the envelope and the encoding rules are defined in different namespaces in order to promote simplicity through modularity.

Web services, as their name implies, are application services offered via the Web. In a typical Web services scenario, a business application sends a request to a service at a given URL using the SOAP protocol over HTTP. The service receives the request, processes it, and returns a response. One of the most common examples is that of a stock quote service, in which the request asks for the current price of a specified stock, and the response gives the stock price. This is one of the simplest forms of a Web service in that the request is filled almost immediately, with the request and response being parts of the same method call. Web services and consumers of Web services are typically businesses,

making Web services predominantly business-to-business (B-to-B) transactions. Although .NET and other initiatives are designed to provide server-resident applications to user clients, an enterprise can be the provider of Web services and also the consumer of other Web services.

4. CORBA

CORBA, or Common Object Request Broker Architecture, is a standard architecture for distributed object systems developed by the Object Management Group (OMG) consortium. The OMG is responsible for defining CORBA. The OMG comprises over 700 companies and organizations, including almost all the major vendors and developers of distributed object technology, including platform, database, and application vendors as well as software tool and corporate developers. CORBA allows a distributed, heterogeneous collection of objects to interoperate. The basic CORBA paradigm is a request for services of a distributed object. Everything else defined by the OMG is in terms of this basic paradigm. The services that an object provides are given by its interface. Interfaces are defined in OMG's Interface Definition Language (IDL). Distributed objects are identified by object references, which are typed by IDL interfaces.

The Object Request Broker (ORB) is the distributed service that implements the request to the remote object. It locates the remote object on the network, communicates the request to the object, waits for the results, and when available communicates those results back to the client. The client and the CORBA object regardless of where the object is located use exactly the same request mechanism. It might be in the same process with the client, somewhere in the same network enterprise or in another country. There is no difference for the client and the architecture in total. Furthermore, the client issuing the request can be written in a different programming language from the implementation of the CORBA object. The ORB does the necessary translation between programming languages. Language bindings are defined for all popular programming languages.

In conclusion, we can say that CORBA objects differ from typical programming language objects in three main ways: a) they can be located anywhere on a network b) they can interoperate with objects written on other platforms, and c) they can be written in any programming language (Java, C++, COBOL, etc) for which there is a mapping from IDL to that language.

B. SURVEY OF SECURITY ISSUES IN RELATED TECHNOLOGIES AND RESEARCH LITERATURE SURVEY

1. WS-Security

Many experts agree that Web Services (WS) lack of security standards is one of the major factors that has slowed the widespread acceptance and implementation of Web Services. As a consequence of this opinion, in April 2002, IBM, Microsoft, and VeriSign published a new Web Services security specification, WS-Security. The specification aims to help enterprises build secure Web Services and applications based on them that are broadly interoperable. Eventually, this specification would be submitted for consideration as a standard, and looking at the amount of commitment that IBM, Microsoft, and VeriSign have invested in it, it may soon go that way. This specification proposes a standard set of SOAP extensions that can be used when building secure Web Services to implement integrity and confidentiality.

WS-Security supports, integrates, and unifies several popular security models, mechanisms, and technologies. This allows a wide array of existing systems to interoperate in a platform- and language-neutral manner in the context of present day Web Services. It also defines a standard set of SOAP extensions. These message headers can be used to implement integrity and confidentiality in Web Services applications. This specification also provides standard mechanisms for Web Services applications to exchange secure, signed messages. Another important factor of WS-Security is that it is a solid, open-standards-based security model and hence will be developed rapidly.

Microsoft and IBM have produced a road map outlining several Web Services security specifications, which is available at <http://www-106.ibm.com/developerworks/security/library/ws-secmap/>. This road map is based on a radical approach to security and defines additional, related Web Services security capabilities within the framework established by the WS-Security specification. By using this framework, enterprises can incorporate the new specifications, as needed, into the different levels of their Web Services applications. The other proposed specifications include WS-Policy, WS-Trust, WS-Privacy, WS-Secure Conversation, WS-Federation, and WS-Authorization. A summary with a brief description of all the above specifications is shown in the Table 1.

Specification	Description
WS-Security	Describes how to attach signature and encryption headers to SOAP messages. In addition, it describes how to attach security tokens, including binary security tokens such as X.509 certificates and Kerberos tickets, to messages.
WS-Policy	Describes the capabilities and constraints of the security (and other business) policies on intermediaries and endpoints (e.g. required security tokens, supported encryption algorithms, privacy rules).
WS-Trust	Describes a framework for trust models that enables Web services to interoperate securely.
WS-Privacy	Describes a model for how Web services and requesters state subject privacy preferences and organizational privacy practice statements..
WS-SecureConversation	Describes how to manage and authenticate message exchanges between parties including security context exchange and to establish and derive session keys.
WS-Federation	Describes how to manage and broker the trust relationships in a heterogeneous federated environment including support for federated identities.
WS-Authorization	Describes how to manage authorization data and authorization policies.

Figure 4. WS Security Summary

Among the main advantages of WS Security is that it is the most comprehensive and elaborate attempt to add security to Web Services. It insulates development teams from the low level specific details of the technologies involved in implementing security.

It also facilitates a rapid change of implementation between technologies without disturbing the existing interfaces between systems. WS-Security is flexible and is designed to be used as the basis for the construction of a wide variety of security models including PKI, Kerberos, and SSL. Specifically, WS-Security supports multiple security tokens, trust domains, signature formats, and encryption technologies. WS-Security is a building block that can be used in conjunction with other Web Services extensions and higher-level application-specific protocols to accommodate a wide variety of security models and encryption technologies.

2. Microsoft's .NET Framework Security

a. Introduction

Microsoft's .NET Framework provides a rich security system, capable of confining code to run in tightly constrained, administrator-defined security contexts. In many of the existing security models, security attributes are assigned to either users and their groups or both. This means that users, and all code run on behalf of these users, are either permitted or not permitted to perform operations on critical resources. This is a common security model in most operating systems. The .NET Framework provides, in a similar way, a security model called role-based security that can be defined by the developer. Another feature that the .NET Framework can also provide is security on code. This is referred to as code access security or sometimes as evidence-based security. With code access security, a user may be trusted to access a resource, but if the code the user executes is not trusted, then access to the resource will be denied. Security based on code, as opposed to specific users, is a fundamental facility to permit security to be expressed on mobile code. Mobile code may be downloaded and executed by any number of users all of which are unknown at development time. The .NET Framework security system functions over the traditional operating system's security, adding a second more expressive and extensible level to operating system security. Both layers complement each other and sometimes the operating system security system can delegate some responsibility to the common language runtime security system for managed code. This is a powerful capability as the runtime security system is finer grain and more configurable than traditional operating system security.

b. Code Access Security

Code access security assigns permissions to assemblies based on assembly evidence. Code access security uses the location from which executable code is obtained and other information about the identity of code as a primary factor in determining what resources the code should have access to. This information about the identity of an assembly is called evidence. Whenever an assembly is loaded into the runtime for execution, the hosting environment attaches a number of pieces of evidence to the assembly. It is the responsibility of the code access security system in the runtime to map this evidence into a set of permissions, which will determine what access this code has to a number of resources, such as the registry or the file system. The default code access security policy has been designed to be as secure as it can be, for most application scenarios of managed code. There are many limitations regarding what non-trusted code from the Internet or local intranet is capable of doing when executed on the local machine. The code access security default policy model thus represents a conservative approach to security, so administrators need to take explicit action to make the system less secure. The Code Access Security is focused on three core abstractions: permissions, evidence, and policies. The security abstractions for role-based security and code access security are represented as types in the .NET Framework Class Library and are user-extendable.

Permissions represent authorization to perform a protected operation. These operations often involve access to a specific resource. In general, the operation can involve accessing resources such as files, the registry, the network, the user interface, or the execution environment. An example of a permission that does not involve a tangible resource is the ability to skip verification. Based on the evidence presented to the security system at assembly load time, the security system grants a permission set that represents authority to access various protected resources. Conversely, resources are protected by a permission demand that triggers a security check to see that a specific permission has been granted to all callers of the resource; if the demand fails, an exception is raised.

Whenever an assembly is loaded into the runtime, the hosting environment presents the security system with evidence for the assembly. Evidence constitutes the input to the code access security policy system that determines what permissions an

assembly may receive. Many of the classes that ship with the .NET Framework, such as Zone, URL, Hash, Site, Application Directory, are used as standard forms of evidence in the security system. The procedure of determining the actual set of granted permissions to an assembly has three main parts:

- A. Individual policy levels evaluate the evidence of an assembly and generate a policy level specific granted set of permissions.
- B. The permission sets calculated for each policy level are intersected with each other.
- C. The resulting permission set is compared with the set of permissions the assembly declared necessary to run, or refuses and the permissions grant is modified accordingly.

During security evaluation, other assemblies might need to be loaded to be used in the policy evaluation process. For example, an assembly can contain a user-defined permission class as part of a permission set handed out by a code development group. Of course, the assembly containing the custom permission also needs to be evaluated. If the assembly of the custom permission is granted the permission set containing the custom permission it itself implements, then a circular dependency ensues. To avoid this, each policy level contains a list of trusted assemblies that it needs for policy evaluation. The list of required assemblies is naturally referred to as the list of "Policy Assemblies," and contains the transitive closure of all assembly required to implement security policy at that policy level. Policy evaluation for all assemblies contained in that list is short circuited to avoid the occurrence of circular dependencies.

*c. **Role-based Security***

The basic component of the code access security system, as it is described in the previous chapter, is the identity of code. However, there is still a need to be able to express security settings based on user identities. The runtime security system also includes role-based security features, which are similar to the implementation of security in many current operating systems. Two core abstractions in role-based security are Identity and Principal. Identity represents the user on whose behalf the code is executing. It is important to remember that this could be a logical user as defined by the application

or developer and not necessarily the user as seen by the operating system. A Principal represents the abstraction of a user and the roles in which a user belongs. Trying to explain better the distinction between the identity and the principal, it can be said that identity is the user from his code point of view and principal is the user from his assigned roles point of view.

3. Java Security

a. Introduction

Since the inception of Java technology, there has been strong and growing interest around the security of the Java platform as well as new security issues raised by the deployment of Java technology. One of Java's main features is its ability to move code over a network and to run that code. Unlike other languages, Java has been designed to do this securely. The first versions of Java security, which used the concept of the “sandbox,” was proved inadequate to support the demands for fine-grained security that can be easily implemented. Recent releases, such as Java 2 Runtime Environment, provide fine-grained security features that enable implementation of a flexible policy decoupled from the implementation mechanism.

Although enforcement of policies during code execution is a substantial part of security, proper security starts at the very beginning, during the generation of byte code. A language's type safety, which is enforced by the compiler and checked by the runtime environment, proves critical to an overall secure environment. There have been many computer security breaches that stemmed from the ability to overflow buffers easily or to access memory unimpeded. Those situations are caused in part by a language's poor type safety and inadequate enforcement in the executing environment. Despite the safety checks enforced by the compiler, the VM must still be able to deal with faulty byte code, whether generated accidentally or maliciously. Java security manifests itself in the following forms:

- a) Protection built into the language,
- b) Building blocks for a flexible secure environment, and
- c) Protection against accidental or malicious attacks to the language and platform.

b. Basic Concepts

A fundamental concept and important building block of system security is the protection domain [Saltzer and Schroeder] [1]. A domain can be scoped by the set of objects that are currently directly accessible by a principal, where a principal is an entity in the computer system to which permissions (and as a result, accountability) are granted. The sandbox concept utilized in JDK 1.0 is one example of a protection domain with a fixed boundary. The protection domain concept serves as a convenient mechanism for grouping and isolation between units of protection. For example, it is possible (but not yet provided as a built-in feature) to prevent protection domains from interacting with each other so that any permitted interaction must be either through trusted system code or explicitly allowed by the domains concerned. Protection domains generally fall into two distinct categories: system domain and application domain. It is important that all protected external resources, such as the file system, the networking facility, and the screen and keyboard, be accessible only via entities within the system domain.

A domain conceptually encloses a set of classes whose instances are granted the same set of permissions. Protection domains are determined by the policy currently in effect. The Java application environment maintains a mapping from code (classes and instances) to their protection domains and then to their permissions. A thread of execution (which is often, but not necessarily tied to, a single Java thread, which in turn is not necessarily tied to the thread concept of the underlying operation system) may occur completely within a single protection domain or may involve an application domain and also the system domain. For example, an application that prints a message out will have to interact with the system domain that is the only access point to an output stream. In this case, it is crucial that the application domain does not gain additional permissions by calling the system domain. Otherwise, serious security implications can ensue. In the reverse situation where a system domain invokes a method from an application domain, such as when the AWT system domain calls an applet's paint method to display the applet, it is again crucial that at any time the effective access rights are the same as the current rights enabled in the application domain. In other words, a less "powerful" domain cannot gain additional permissions as a result of calling or being called by a more powerful domain.

c. New Enhancements of the Java 2 SDK

There are many enhancements in the security architecture of the latest Java 2 Standard Development Kit (SDK). Several features that were previously available separately are now part of the core API set. These include support for encryption and decryption with the Java Cryptography Extension (JCE), support for Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols with the Java Secure Socket Extension (JSSE), and support for user -based authentication and access controls with the Java Authentication and Authorization Service (JAAS). In addition to the inclusion of these previously optional packages, we can find new support for building and verifying certificate chains with the Java Certification Path API and support for the Kerberos V5 mechanism under Java GSS-API and JAAS. Additional enhancements were made in improving the security policy-managing tool, `policy tool`, and in adding support for dynamically loading security policies.

The Java Cryptography Extension (JCE) provides support for encryption, decryption, key agreement, Message Authentication Code (MAC), and some other cryptographic services. Due to import control restrictions of some countries, the JCE jurisdiction policy files shipped with the Java 2 SDK, release 1.4 allow "strong" but limited cryptography to be used. An "unlimited strength" version of these files indicating no restrictions on cryptographic strengths is available for those living in eligible countries.

The Java Secure Socket Extension (JSSE) library provides support for communicating using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. Where the JCE operates on specific local data structures, the JSSE uses a different abstraction, applying encryption/decryption to network socket traffic. It adds server authentication, message integrity, and optional client authentication. Most people think of SSL and TLS as the secure HTTP protocol, better known as HTTPS. SSL (and thus HTTPS) permits encrypted traffic to be exchanged between the client and server. For example, in an SSL mode, after an SSL client initiates a conversation with an SSL server, the server sends an X.509 certificate back to the client for authentication (SSL also supports mutual authentication). The client then checks the validity of the certificate. Assuming the server is verified, the client generates a pre-master secret key, encrypts it

with the server's public key from the certificate, and sends the encrypted key back to the server. From this pre-master key, the client and server generate a master key for the session. After some basic handshaking, the encrypted exchange can commence.

The Java Authentication and Authorization Service (JAAS) provides for the authentication of users and the authorization of tasks based upon that authentication. This is an enhancement to the prior standard security model capabilities of enabling a specific set of tasks based on authentication. Previously, anyone authenticated had access to the same security restrictions. Now, there is control on what tasks are available for a specific authenticated user.

The Java GSS-API (Generic Security Service) adds Kerberos support to the Java platform. Kerberos is a network authentication protocol, originated at the Massachusetts Institute of Technology (MIT) as project Athena back in 1987. The Java GSS-API offers single sign-on within a domain, if everything within the domain has been Kerberos-enabled. Support is also provided for single sign-on across different security realms over a network. Used in conjunction with JAAS, once a user's identity is established, future authentication requests are no longer necessary.

The fifth of the libraries, which is now standard, is the Java Certification Path API. It provides classes for building and validating certificate chains, an important requirement of a Public Key Infrastructure (PKI). These certificates provide for the storage of security keys for users. By trusting the issuer of a certificate that holds the keys, and trusting the issuer of the certificate that trusts the original certificate, you establish chains of trust. By following this certificate path chain, you eventually either end up with a certificate issued by a Certification Authority (CA) that you trust or a certificate issued by a CA that you do not trust. Thus, the relying party can ensure a subject's public key is genuine and trusted based on the trustworthiness of the underlying certificate chain.

4. COM & DCOM

a. Introduction

The Component Object Model (COM) refers to both a specification and implementation developed by Microsoft Corporation, which provides a framework for

integrating components. This framework supports *interoperability* and *reusability* of distributed objects by allowing developers to build systems by assembling reusable components from different vendors that communicate via COM. It also defines an application-programming interface (API) to allow for the creation of components to integrate custom applications or to allow diverse components to interact. However, in order to interact, components must adhere to a binary structure specified by Microsoft. As long as components adhere to this binary structure, components written in different languages can interoperate independently of the programming language or the platforms (MS Windows, UNIX, Macintosh). In addition, COM provides mechanisms for the following:

- ✓ Communications between components, even across process and network boundaries
- ✓ Shared memory management between components
- ✓ Error and status reporting
- ✓ Dynamic loading of components

Distributed COM (DCOM) is an extension to COM that allows network-based component interaction. While COM processes can run on the same machine but in different address spaces, the DCOM extension allows processes to be spread across a network. With DCOM, components operating on a variety of platforms can interact, as long as DCOM is available within the environment. It is best to consider COM and DCOM as a single technology that provides a range of services for component interaction, from services promoting component integration on a single platform, to component interaction across heterogeneous networks. In fact, COM and its DCOM extensions are merged into a single runtime. This single runtime provides both local and remote access.

b. Security

The Component Object Model (COM) can make distributed applications secure without any security-specific coding or design in either the client or the component. Just as the COM programming model hides a component's location, it also hides the security requirements of a component. The same binary code that works in a

single-machine environment, in which security may be of no concern, can be used securely in a distributed environment. COM provides two distinguishable categories of security. The first is termed activation security, and it controls which objects a client is allowed to instantiate. The second form is call security, which dictates how security operates at the per-call level on an established connection from a client to a server object.

Activation security controls which classes a client is allowed to launch and to retrieve objects from. The Service Control Manager of a particular machine automatically applies activation security. Upon receipt of a request from a remote client to activate an object the Service Control Manager of the machine checks the request against the following information stored within its registry:

- ✓ Machine-wide settings for securing activation
- ✓ Per-class settings for activation

Call security in COM is provided via two mechanisms in order to secure calls. The first is similar to DCE RPC, which means that COM provides functions and interfaces that applications can use to do their own security checking. COM runs the second mechanism automatically. If the application provides some setup information, COM will make all the necessary checks to secure the application's objects. This automatic mechanism does security checking for the process, not for individual objects or methods. Applications requiring more fine-grained security can perform their own security checking. The two mechanisms are not exclusive: an application can ask COM to perform automatic security checking and also perform its own. COM call-security services are divided into three categories: general functions called by both clients and servers, new interfaces on client proxies, and server-side functions and call-context interfaces. The general functions initialize the automatic security mechanism and register authentication services. The proxy interfaces allow the client to control security on calls to individual interfaces. The server functions and interfaces allow the server to retrieve security information about a call and to impersonate the caller.

C. IC METADATA STANDARD FOR PUBLICATIONS

The IC Metadata Sub-Working Group (MSWG) has developed the Intelligence Community Markup Language (ICML), which is based on a number of data modeling

activities that have occurred in the IC over the last few years. The ICML standard is being developed in response to requests by numerous organizations within the IC to have an IC-wide mandated XML model to support interoperability of intelligence content across producers and consumers of information within the IC. Based on XML, ICML defines tags that communicate important additional information about intelligence content. Furthermore, ICML introduces:

- 1) Many document structures, such as reports, articles, and analytical packets.
- 2) An expanded collection of document metadata separated into administrative and descriptive categories.
- 3) The most commonly used generic document components, such as paragraphs, lists, tables, and media.
- 4) CAPCO-compliant security labels.
- 5) Descriptive content tags indicating better the subject matter of the information.

ICML first appeared at the Intelink Conference in September 2000 in the form of an XML Document Type Definition (DTD). As stated in that early version, the purpose of that DTD was

to provide a common set of XML elements (TAGS) for implementing security-based metadata throughout the IC. This DTD may be incorporated into various organizational XML-based DTDs by calling the entity declaration referenced within the core IC Security DTD.

At the end of 2001, the Intelligence Community Metadata Standard for Publications (IC-MSP) was published, as part of the IC CIO Executive Council and Working Group commitment to IC inter-organization interoperability. Since the first version of IC-MSP (IC-MSP Release 0.5, 12/08/01), the focus remains to aid finished intelligence production. And because the majority of the intelligence content being produced within the IC is in the form of documents, the IC-MSP Panel decided that limiting the scope of the initial IC-MSP release to document type of intelligence content would yield the most benefit within the shortest period of time. The IC-MSP standard is a flexible markup application can support many processes such as authoring, storage, paper or web output etc. Furthermore, the IC-MSP consist of XML elements, structures

or models that are either characteristic to IC analyses and products or have general application and have been established by government and industry as official or de facto standards.

During the past year, six versions of the above IC-MSP were published, in order to implement the improvements and the feedback of the agencies. The latest version 1.0a was published on 16th of September 2002. It is actually the same with its previous version 1.0 (5th July 2002) containing only some corrections to the Data Element Dictionary (DED) in order to remove disconnects between the DED and the various DTD modules. At the same time, due to the increased requirements for security, The Intelligence Community Metadata Standards for Information Assurance (MSIA) are being developed to ensure the security of XML-based transactions throughout the community. The IC MSIA includes additional efforts to facilitate secure transactions within the Intelligence Community such as:

- Digital Signatures (XML-DSig)
- Security/Encryption (XML-Sec)
- Key Management System (XML-KMS)
- Information Security Marking (XML-ISM)

III. DBMS ARCHITECTURES.

A. OVERVIEW OF THE BASIC SECURE DBMS

1. Historical Background

In the mid 1970s, the US Air Force sponsored two programs for trusted relational Database Management System (DBMS) research. The first work, done by Hinke and Schaefer in 1975, [6] documented the design for a high assurance DBMS. It was based on the Multics operating system, which contained all the relevant trusted code, and therefore was responsible for all the access control. One year later, I. P. Sharp and Associates [5] developed a model for a multilevel relational DBMS. This model was based on a layered internal DBMS architecture using a method introduced by Parnas. Both the above architectures depended on the security of the underlying operating system's kernel, and for that reason they were referred to as "kernelized architectures."

Two years later, the US Navy sponsored two more efforts contributing to the trusted DBMS development. The first, called "The Military Message System Model," introduced a multilevel container that holds sub-objects that the container's level dominates. Even though it influenced many trusted DBMS designs, it did not result in the development of a design for a multilevel DBMS. The second program sponsored by the Navy intended to address the specific requirements for Navy surveillance systems. Among the special features of this model was the usage of the container concept and nesting database objects. Both the "Integrity-prototype" and the "Trudata" DBMS used the Navy model as their basis.

One of the most important points in the trusted DBMS development occurred when the Air Force sponsored a study on trusted data management at Woods Hole, Massachusetts. One of the groups there recommended a separation into three approaches: the kernelized (or Hinke-Schaefer) approach, the distributed (or back-end or replicated) approach, and the cryptographic integrity-lock (or spray painting) approach. All of those three approaches form the first of the two main categories of the multi-level secure DBMS architectures, which was named "the Woods Hole architectures." The other main category is referred to as "the trusted subject DBMS architectures," which mainly differs

from the Woods Hole architectures in that they perform their own mandatory access control. Since then several variations have developed within these two main categories.

2. Woods Hole Architectures

The Woods Hole architectures assume that an untrusted, usually commercial-off-the-shelf (COTS) DBMS is used to access the data. In order to provide an overall secure DBMS system, trusted code is developed around that DBMS. The three different Woods Hole architectures address three different ways to wrap code around the untrusted DBMS.

The Kernelized architecture scheme (Figure 5) uses a trusted operating system and multiple copies of the DBMS; each is associated with a trusted front end. The trusted

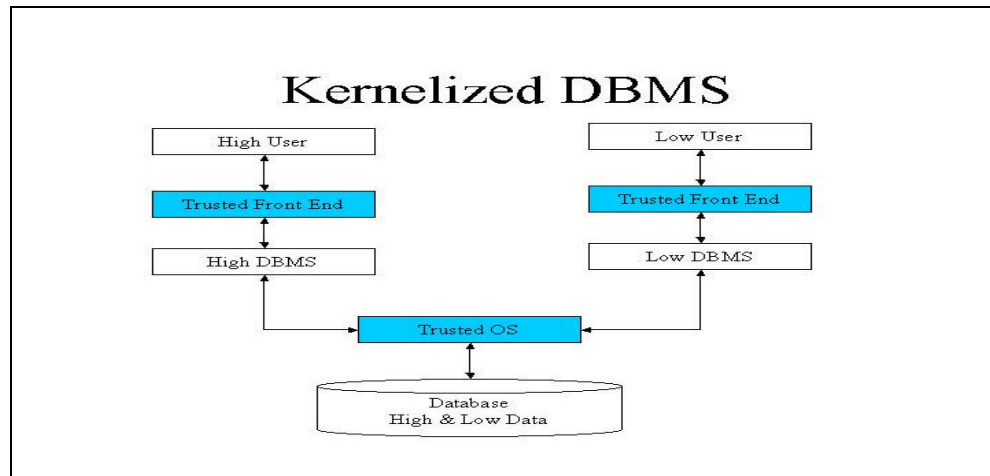


Figure 5. Kernelized DBMS

front end-DBMS pair is associated with a particular security level. Between the DBMS and the database, a portion of the trusted operating system keeps the data separated by security level. Each trusted front end is trusted to supply requests to the proper DBMS. The database is separated by the security level. The trusted operating system separates the data when it is added to the database by a DBMS and combines the data when it is retrieved (if allowed by the security rules of the requesting DBMS). The high DBMS obtains data combined from the high and low segments of the database. The low DBMS can only obtain data from the low segment of the database. A benefit of this scheme is that the access control and the separation of data at different classification levels is

performed by a trusted operating system rather than the DBMS. Data at different security levels is isolated in the database, which allows for higher-level assurance. Users interact with a DBMS at the user's single-session level.

The distributed architecture scheme (Figure 6) uses multiple copies of the trusted front end and DBMS, each associated with its own database storage. In this architecture

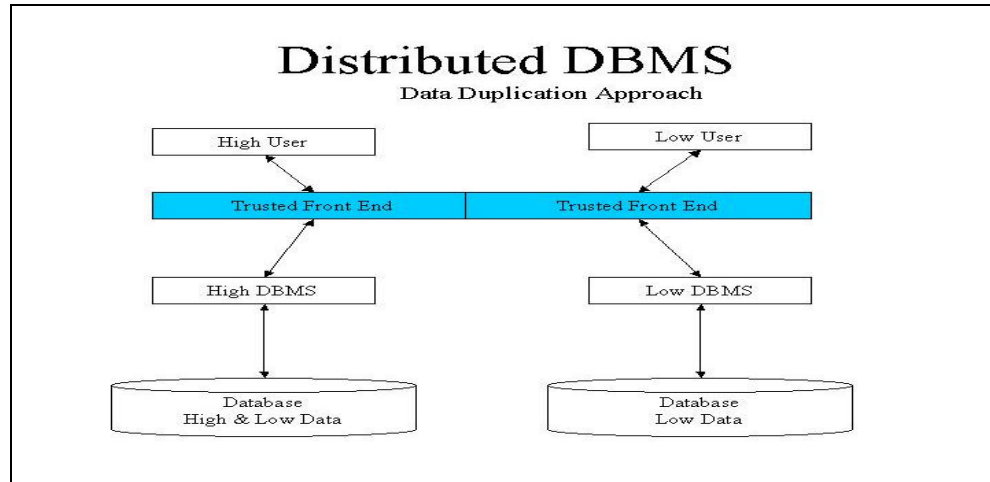


Figure 6. Distributed DBMS

scheme, low data is replicated in the high database. When data is retrieved, the DBMS retrieves it only from its own database. A benefit of this architecture is that data is physically separated into separate hardware databases. Since separate replicated databases are used for each security level, the front end does not need to decompose user query data to different DBMSs.

The Integrity Lock architecture scheme (Figure 7) places a trusted front-end filter between the users and the DBMS. The filter provides security for the MLS. When data is added to the database, the trusted front-end filter adds an encrypted integrity lock to each unit of data added to the database. The lock is viewed by the DBMS as just another element in the unit stored by the DBMS. The encrypted lock is used to assure that the retrieved data has not been tampered with and contains the security label of the data. When data is retrieved, the filter decrypts the lock to determine if the data can be returned to the requester. The filter is designed and trusted to keep users separate and to store and

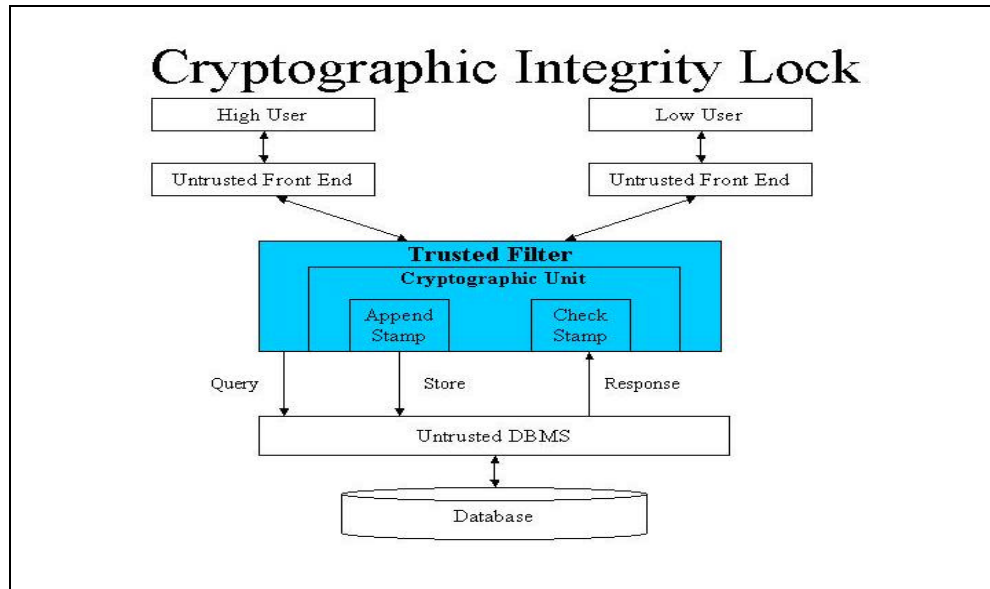


Figure 7. Cryptographic Integrity Lock DBMS

provide data appropriate to the user. A benefit of this scheme is that an untrusted COTS DBMS can perform most indexed data storage and retrieval.

3. Trusted Subject Architectures

The Trusted Subject architecture (Figure 8) is a scheme that contains a trusted DBMS and an operating system. The DBMS can be customized with all the required security policy (the security rules that must be enforced) developed in the DBMS itself. The DBMS uses the associated trusted operating system to make actual disk data accesses. This is the traditional way of developing MLS DBMS capabilities, and it can achieve high mandatory assurance for a particular security policy by sacrificing of some DBMS functionality. This scheme results in a special purpose DBMS and operating system that requires a large amount of trusted code to be developed and verified along with the normal DBMS features. The trusted code provides security functionality and has been designed and developed using a rigorous process, tested, and protected from tampering in a manner that ensures the Designated Approving Authority (DAA) that it performs the security functions correctly. The DAA is the security official with the authority to say a system is secure and thus its use is permitted. A benefit of the trusted subject architecture is that the DBMS has access to all levels of data at the same time,

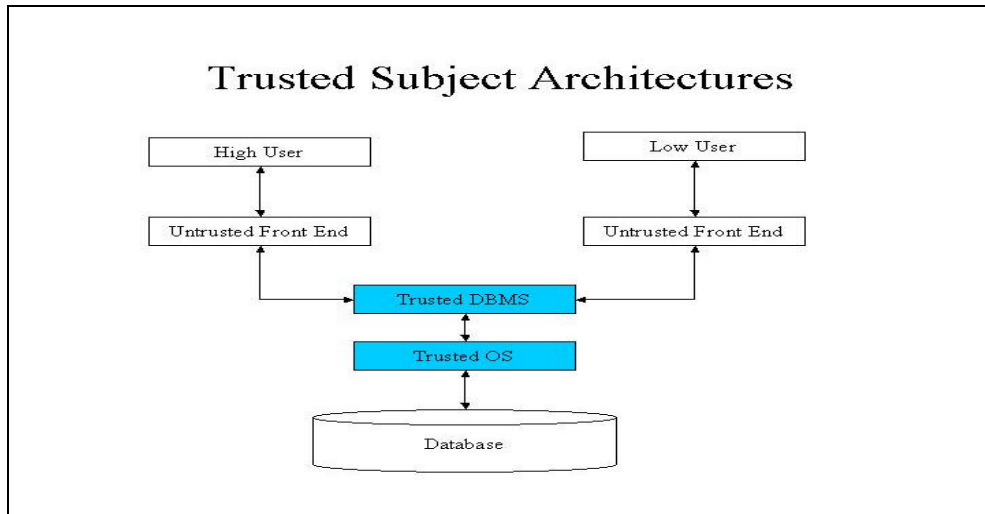


Figure 8. Trusted Subject Architectures

which requires less processing time when retrieving or updating data. This scheme can also handle a wide range of sensitivity labels and can support complex access control. A sensitivity label identifies the classification level (e.g., confidential, secret) and a set of categories or compartments that apply to the data associated with the label.

B. ANALYSIS AND REQUIREMENTS OF DBMS ARCHITECTURES

1. Basic Requirements

The DBMS that will manage a large amount of documents, like those in large organization must, at least, include a number of capabilities that will assure an acceptable level of functionality. First of all, the DBMS has to be *scalable*, in both performance capacity and incremental data volume growth. The proposed solution scales in a near-linear fashion and behaves consistently as the database increases in size, the number of concurrent users and the complexity of queries. DBMS must also have a *powerful* design in order to support complex decisions with multi-users and a mixed workload. The optimizer should be mature enough to support every type of query with good performance, and must also determine the best execution plan based on the changing data demographics. The optimizer must also check on the conditional parallelism and determine what causes the variations in the parallelism deployed. Finally, it must check on the dynamic and controllable prioritization of resources for queries.

Manageability through minimal System Administrator intervention is another feature that must be present in DBMS architecture. System administration can be simpler if the DBMS provides a single point of control, which will allow one to create and implement the new tables at any time. In order to support the critical applications of a large-scale organization's mission, the DBMS must have a high *availability* level. Any down time and any issues that might deny or degrade service to end-users must be completely transparent to the system administrators. These down time requirements could include batch load times, software/hardware upgrades, severe system performance issues and system maintenance outages.

The design and the system architecture should also be *flexible* and *extensible* to keep pace with evolving business requirements and to improve the value of the existing investment in hardware and applications. The impact of repartitioning tables and the addition or deletion of columns must be minimal. The design should also provide optimal performance across the full range of normalized, star and hybrid data schemas with large numbers of tables.

Nowadays, *interoperability* through the web or internal networks has also become a major factor when deciding on a DBMS. Its ability to support multiple applications from different business units, leveraging data that is integrated across business functions and subject areas is considered a critical feature of the DBMS. All the above features, of course, must be previously proven, for the specific product to avoid a risk.

As more business is conducted on the Web, securing data in motion and user identities is a growing concern. User management and deploying secure infrastructures have risen to the top of the database administrator's priorities. The architecture of the DBMS should protect the data stored in the database, when transferred from unauthorized access, and from malicious destruction or alteration and accidental introduction of inconsistency. Encryption forms the basis for secure authentication of users. Even though absolute protection of the database from malicious abuse is not possible, a sufficient increase in the required cost to the perpetrator can be a deterrent. In order for the DBMS to be protected against malicious or unauthorized access, several forms of authentication and authorization must be enforced. The system should also allow the users to grant some

forms of authority to other users ensuring at the same time that this authorization can be revoked at some future time. Roles help to assign specific sets of privileges to different groups of users inside an organization. When the stored data are highly sensitive, the various authorizations provided in the database may not be sufficient. In such cases, data must be encrypted. Only the user who knows how to decrypt and possesses the necessary decryption key should be allowed to read the data. Compatibility with the modern techniques of encryption-decryption, Public Key Infrastructure (PKI), and Digital Signatures are highly required when deciding on a DBMS for today or for the future world.

2. XML and DBMS Architectures

Extensible Markup Language (XML) is emerging as the format of choice for a variety of types of data, especially documents. Providing its ability to tag different fields, XML makes searching simpler and more dynamic. It is also ideal for organizations trying to meld incompatible systems because it can serve as a common transport technology for moving data around in a system-neutral format. In addition, XML can handle all kinds of data, including text, images and sound, and is user-extensible to handle anything special. One of the main concerns until now has been how to manage the XML-tagged data. A suitable and convenient solution is to use databases to store, to retrieve and to manipulate XML. The idea is to place the XML-tagged data in a framework where searching, analysis, updating and output can proceed in a more manageable, systematic and well-understood environment. The primary advantage of databases is that users are familiar with them and their behavior, so combining XML with a database context seems natural.

A few different approaches exist regarding the use of XML in a database. The main categorization is according to the format that the DBMS uses to store the XML document. Half of those categories store XML in its native format and the other half transform it and store it in a common relational or object-oriented database. There are a number of reasons to use existing database types and existing database products to store XML, even if it is not in its native form. First, ordinary relational and object-oriented databases are well known, while native XML databases are new. Second, as a result of familiarity with relational and object-oriented databases, users understand their behavior, especially regarding performance. Many businesses are reluctant to move to a native

XML database whose characteristics—especially scalability—have not been tested. Finally, relational and object-oriented databases are safe choices in the corporate mind. On the other hand, one of XML's attractive features is its hierarchical organization, which database tables crush. Relational databases must map XML to relational tables and therefore flatten XML structures into rows and columns each time data is needed. In addition, translating XML to and from the database requires considerable processing, especially for large or complex documents. This performance factor may be important when dealing with web pages through a dial-up Internet connection, but at the same time this performance factor may be of slight importance if it is used in an Intranet or through a high-speed Internet connection.

Storing XML documents in a relational or object-oriented database can be done in different ways. First, one could extract the data elements in an XML document and store them as data rows and columns in an SQL database. Many call this technique “element storage.” Given an example of an XML intelligence document, a number of SQL tables could be created with columns for the individual elements of document. The “*IdentifierList*,” the “*DocumentID*,” the “*Publisher*,” or the “*AgencyAcronym*” could be some of the columns. Then, those kinds of data can be managed in SQL with normal SQL operations. By retrieving the data, an XML document can be produced and published or transferred via the Web. In case there are new data that must be stored, an “UPDATE” operation, using common SQL commands, must be done. *Element storage* has the advantage that all of the data from the XML document is available to SQL as normal SQL data so that it can be queried and updated with SQL operations. However, element storage has the disadvantage of the extra overhead of assembling and disassembling the XML documents for interchange.

One can also store the XML document in a single SQL column. Since XML is primarily a file format, a natural storage mechanism is simply a flat file. It is usually referred to as “document storage.” This approach has some drawbacks, such as data isolation problems, integrity checks and concurrent-access problems. However the wide availability of XML tools that work on file data makes it relatively easy to access and to query XML data stored in files. Thus, this storage format may be sufficient for some kinds of applications. Using the previous example, a table can be created having a

column for the specific category of IC's document. The data-type of that column could be SQL text, or a Java class designed for XML documents in general, or a Java class designed specifically for each specific type of IC XML document. *Document storage* eliminates the need for assembling and disassembling the data for interchange. However, there is a need to use Java (or any other language) methods to reference or to update the elements of the XML documents while they are in SQL, which could be slower and less convenient than the direct SQL access of element storage.

Finally, there is a combination called "hybrid storage" of the above *element* and *document* approaches that exploits the advantages of both. This mainly stores an XML document in an SQL column, but at the same time extracts some of its data elements into separate columns for faster and more convenient access. Given a previous example, one can create SQL tables, such as for *document storage*, and then one can include (or later add) just one or more columns to contain elements extracted from the same type of documents, such as for *element storage*. *Hybrid storage* balances the advantages of element storage and document storage, but has the cost and complexity of redundant storage of the extracted data.

Since XML documents are character data, we could analyze them and extract their data using SQL character string operations. However this process can be complicated and tedious. Using XML with SQL is greatly facilitated by using XML tools written in Java (or C++,) which are called "parsers," and their main job is to analyze and to validate XML documents. Specifically, XML parsers provide many capabilities, including the following:

- ✓ Checking that a document is well formed and valid;
- ✓ Handling character set issues;
- ✓ Generating a Java representation of a document's parse tree;
- ✓ Building or modifying a document's parse tree;
- ✓ Generating a document's text from its parse tree.

There are many XML parsers available in Java, often with a free license or public domain. Most XML parsers implement two standard interfaces, the Simple API for XML

(SAX) and the Document Object Model (DOM). SAX provides facilities for specifying input sources, character sets, and routines to handle external references. It generates events during the parse so that user routines can process the document incrementally, and it returns a DOM object that is the parse tree of the document. On the other hand, DOM provides facilities for stepping through the parse tree and for assembling a parse tree. Applications that use the implementations of the SAX and DOM interfaces of an XML parser can be portable across XML parsers.

Another area of concern about the XML storage is where the mapping should be done. Almost all the languages that could manipulate XML can run in both the client and the server. For example, Java methods can be executed in either the client or server environment, giving us a choice of which environment to map an XML document to or from SQL. This is a consideration only for element storage and hybrid storage, since document storage involves little or no processing of the document. If the main priority is the efficiency, then mapping should be done in the client and only the SQL data should be transferred between the client and the server. Unfortunately this approach could not provide us with a high level of security. When security is the priority, the entire process must be done in the server where the appropriate security policy can be enforced.

C. RESEARCH ON RELATED DBMS PRODUCTS

1. Oracle 9i

Oracle 9i (where *i* reflects Oracle's emphasis on the Internet) is not only the version name for the database server, but is also the family name for a whole suite of products around the core server. These products include the Oracle9i Application Server, the Oracle9i DBMS and the Oracle9i Developer Suite. It was officially released in Europe at the Oracle Open World in June 2001 and became available immediately thereafter. Oracle 9i brings new or enhanced functionality in many areas such as availability, scalability, performance, manageability, Internet content management and security. Oracle's Advanced Security option can be separated into three major parts:

- Preserving the privacy and integrity of network data and communication.
- Providing strong authentication services for users, databases and web servers
- Enterprise User Management

It is widely known that all network traffic is vulnerable to eavesdropping, data capture, replay, data modification and person-in-the-middle attack. The network security features of Oracle Advanced Security address these concerns by integrating encryption and integrity checking. The cryptographic functionality in Oracle Advanced Security converts all clear text into cipher text. The cipher text is transmitted across the network in a way in which it is computationally unfeasible to convert the cipher text back into its corresponding plain text without the correct key. The algorithms used to encrypt Oracle Net traffic are:

- Triple DES (3DES, 2Key and 3Key 112- and 168-bit keys)
- RC4 (40-, 56-, 128-, 256-)
- Advanced Encryption Standard (AES)

Oracle Advanced Security also provides encryption capabilities to thin Java Database Connectivity (JDBC) clients. All Oracle Net traffic between the database and the client (or a web server) is encrypted. A key benefit of using Oracle Advanced Security encryption is that the users or the database do not need to have digital certificates or communicate over Secure Socket Layer (SSL).

Data integrity protects against data modification and replay attacks and proves to the receiver of the message that the message has not been tampered with in transit. This is one of the major requirements in today's online world. Data integrity checking, which is also called "cryptographic check summing" provides sequencing and hashing to protect against these packet attacks. Oracle Advanced Security provides industry standard implementations of Message Digest 5 (MD5) or the Secure Hashing Algorithm (SHA -1) for providing data integrity.

There is a need for augmenting the password-based authentication with stronger measures to identify the users. Oracle Advanced Security provides several strong authentication schemes while supporting industry standards including Kerberos, Smart cards, Remote Dial-in User Service (RADIUS), Distributed Computing Environment (DCE), Standards-based Public Key Infrastructure (PKI), Entrust Profiles, and X.509v3 compliant digital certificates over Secure Socket Layer (SSL). Strong authentication

mechanisms such as Kerberos, DCE and X.509v3 certificates can also provide Single Sign On capabilities to applications that rely on these authentication services.

Today's business environment requires an around-the-clock user administration just as there is a need for a twenty-four hours, seven-day a week application availability. The costs of user administration could be very high if the users and their authorizations continue to be duplicated across the different applications that are deployed throughout the enterprise. Oracle's Enterprise User Security provides the ability to manage enterprise-wide users easily and securely by

- centralizing the storage of user credentials, roles and privileges in an LDAPv3 compliant directory server;
- providing the infrastructure to enable single sign-on using X.509v3 compliant certificates;
- allowing password authenticated database users to be centrally managed as password authenticated enterprise users.

2. IBM DB2 Universal Database V7

The DB2 Universal Database (UDB) is a family of products that covers a broad range of heterogeneous platforms, scaling from handhelds up to mainframes. One of the most popular products of that family is the DB2 Universal Database for Unix, Windows and OS/2 (DB2 UDB for UWO), which is packaged into four editions

- DB2 UDB Personal Edition for usage in a single-user mode (DB2 UDB PE)
- DB2 UDB Workgroup Edition applications and data shared in a workgroup or department (DB2 UDB WE)
- DB2 UDB Enterprise Edition for complex configurations and large database needs from uniprocessors to the largest SMP's (DB2 UDB EE)
- DB2 UDB Enterprise – Extended Edition for large database support in Massively Parallel Processor (MPP's) or clustered server environments (DB2 UDB EEE)

DB2 UDB 7.2, released in June 2001, is identical to Version 7 FixPak 3 as far as the database engine is concerned. However all of the enhancements outside of the engine (e.g. new connectors, improvements to the warehouse manager) are shipped only as part of V7.2. The Version 7.2 of DB2 focuses on integrating business intelligence functionality, on integrating IBM MQSeries and on integrating XML documents supporting Web Services based on XML/SOAP/UDDI. Moreover, DB2 UDB 7.2 includes many enhancements on store procedures and data management.

To protect data and resources associated with a database server, DB2 uses a combination of external security services and internal access control information. To access a database server one must pass some security checks before being given access to a database or other resources. The authentication of a user is completed using a security facility outside of DB2. The security facility can be part of the operating system, a separate product, or, in certain cases, may not exist at all (like Windows 95). On UNIX platforms, the security facility is in the operating system itself. DCE Security Services is a separate product that provides the security facility for a distributed environment. DB2 uses the security facility to authenticate users in one of two ways. First, DB2 acquires the user's successful security system login as evidence of the user's identity and allows him or her to work with local commands to access local data or to work with remote connections where the server trusts the client authentication. Second, DB2 accepts a user ID and password combination and uses the successful validation of the ID and password by the security facility as evidence of the user's identity. In this way, DB2 allows remote connections in which the server requires proof of authentication and use of operations when the user wants to execute a command under an identity other than the identity used for login.

This authorization process is performed inside the DB2 by using tables and configuration files. Each authorized name is associated with a permissions record. The two types of permissions recorded by DB2 are privileges and authority levels. A privilege defines a single permission for an authorization name, enabling a user to create or access database resources. Privileges are stored in the database catalogs for a given database. Authority levels provide a method of grouping privileges and controlling higher-level database manager maintenance and utility operations. The authorities used by DB2 are

- 1) System Administration Authority (SYSADM)
- 2) Database Administration Authority (DBADM)
- 3) System Control Authority (SYSCTRL)
- 4) System Maintenance Authority (SYSMAINT).

The privileges are

- 1) Database Privileges
- 2) Schema Privileges
- 3) Table and View Privileges
- 4) Package Privileges
- 5) Index Privileges.

Database-specific authorities are stored in the database catalogs for each database while system authorities are stored in the database manager configuration file for a given instance. Moreover, the concept of “groups” provides a useful way of performing authorization for a collection of users without having to grant or to revoke privileges for each user individually. Unless otherwise specified, group authorization names can be used anywhere authorization names are used for authorization purposes. The recorded permissions are compared to the authorization name of an authenticated user and those of groups in which the user is a member. Depending upon the comparison, DB2 decides whether to allow or to deny the user the requested access.

3. Sybase Adaptive Server Enterprise (ASE) 12.5

Sybase Adaptive Server Enterprise (ASE) 12.5 includes a Policy-Based Access Control framework, which provides a powerful and flexible way of protecting data, all the way down to the row level. Security policies can be defined according to the value of individual data elements, and then the server can enforce these policies. This means that once a policy has been defined, it is automatically invoked whenever the affected data is queried, whether through an application, a specific query, a stored procedure, or a view. Because the server enforces security, the security administration of an ASE is simpler. The security administration should focus on defining a security policy to enforce

consistently across the entire server. This is accomplished through the combined capabilities of Access Rules, Login Triggers, and Domain Integrity Rules. Let's examine each of these capabilities.

Access Rules are the fundamental concepts of Policy-Based Access Control. An Access Rule is bound to a specific column and then invoked on any SELECT, UPDATE, or DELETE operation on the corresponding table. Login Triggers are stored procedures that execute as part of the login process. They are a convenient way of configuring an Application Context by looking up and setting values for all of the attributes within a context. "Context" a user-defined context can be used by one or more applications. Contexts are set up on a session-by-session basis allowing the security policy to be based on properties of both the application and the user invoking the application. Furthermore, Login Triggers can query other tables and then use that data to support a number of different account usage policies. Domain Integrity Rules are existing ASE server-enforced integrity mechanisms that can be used in conjunction with Access Rules to provide security policy control over the flow of information into and through the server. As is the case with Access Rules, Domain Integrity Rules are bound to columns. They are invoked on UPDATE and INSERT operations.

Sybase ASE 12.5 offers two methods of authentication: the famous user name and password pair, and the digital certificates. The client passes a user name and password to create an authenticated session with the server. This authentication is accomplished on the server when it places the user name on an Access Control List role using its own management tool. Then the server can either use the user name alone, or it can authenticate the password using the underlying operating system. This method places all enterprise-class authentication verification into a single repository—the operating system, which according to Sybase heightens security. By using the enterprise's operating system as the password repository, the involved "links" are less and the chance that security chain will break is accordingly smaller.

The other method of authentication provided by ASE is the X.509 digital certificates. The certificate contains information, such as the user's name and the authority that issues the certificate, and then verified the users identity and the user's

public key. Digital certificates add an extra level of security over traditional user name and password pairs by controlling the possession of the certificate. The Server's Security Manager accepts a user name and allows access to a role, as well as allows the user to provide a digital certificate by choosing from a list of known digital certificates. In this way, no user names or passwords are ever sent across the wire. The Server's Security Manager can also accept digital IDs from the Entrust PKI to authenticate users. The Entrust PKI allows advanced features like single login for every application within the enterprise. The Entrust ID can be used the same way every other digital certificates are used, utilizing Entrust's PKI as the certificate database.

After the authentication process, the next step is to control the objects to which the authenticated users have access. Sybase ASE uses Access Control Lists (ACL) to manage this type of object access. An Access Control List (ACL) works with the idea of building a "role" and placing specific users in that defined role. Each role would have different privileges according to that role's responsibilities and tasks that must be performed. A new role can be defined through the Server's Security Manager. As soon as an administrator has defined that role, he can open it and can configure the authorized users and digital certificates and the unauthorized users and digital certificates. When a role is created and the users are placed into that role, then this role can be applied directly from within the manager. This allows one to choose which level of granularity best fits each business requirements. When a user's business role changes within the organization, the administrator can simply move that user into a different role in the server, and all of their access controls will migrate with them. As components are reused in future applications, the same security restrictions can also be reused.

D. THREE-TIER ARCHITECTURE

Many of today's applications can be divided into three distinct areas:

- i. *The Presentation Logic*: the user interface that displays data to the user or accepts input from the user.
- ii. *The Business Logic*: validates and processes the data, ensuring that it is consistent and in accordance to the requirements and the specifications before being added to the database.

- iii. *The Data Access Logic*: communicates with the database and provides access to the tables and indices. It also packs and unpacks the data.

The user interface in those applications is usually an HTML, XML or XHTML file, either dynamically generated for each case, or a saved, static one. Furthermore, the front-tier, user interface can also contain client-side scripts and sometimes Java applets. But when portability and interoperability is the main concern, the XHTML is the preferred mechanism for the user's representation. Almost all the browsers support XHTML; so designing the user interface to be accessible through a Web browser guarantees a portability and interoperability across most platforms. The user interface communicates with the middle-tier business logic by using the networking features that the browser provides automatically.

When the middle-tier receives a request from the user interface, it processes the request according to the business logic and then accesses the database to manipulate any data required. Generally, in today's multi-tier architectures, Web servers are increasingly used to build the middle tier. They can efficiently provide the business logic that manipulates data from the databases and that also communicates with the client Web browsers. This request-response model of communication between the client-browser and the server is accomplished by using a specific Java programming called a servlet. A servlet extends the functionality of a server and is based on the *javax.servlet* and *javax.servlet.http* packages that provide the necessary classes and interfaces.

The middle-tier servlet interacts with most of the third-tier database systems through the Java Database Connectivity (JDBC), which provides all the means required for communication. Developers need not be familiar with the specifics of each database system. They use common SQL-based queries and the JDBC driver handles the specifics of interacting with each database system. Moreover, other technologies like ODBC, developed by Microsoft, also provide generic access to disparate database systems on the Windows platform (and some UNIX platforms). In those cases, Java enables the JDBC - to-ODBC driver to allow any Java program to access any ODBC data source.

Organizations that divide their applications according to the above three distinct and separate areas can gain many advantages. For example, component roles are

specialized, improving maintainability; networking and I/O overheads. They are also clearly defined within a 3-tier framework, which provides a good basis for component-based development and reusability. Components in the business layer can be shared by any number of components in the presentation layer. Furthermore, using a 3-tier architecture enhances infrastructure independence. This is because presentation and data access areas that are often infrastructure-dependent are separated from the application's business logic. Finally, a specific set of skills is required to develop each tier, so tiers can be developed independently of one another. For example, the thin presentation tier allows front-end experts to do their work without being affected by developments occurring in the business logic tier.

IV. IMPLEMENTATION

A. SECURITY ARCHITECTURE OF THE PROTOTYPE DEVELOPMENT

1. Introduction

The prototype is based on the three-tier architecture, which discussed in the previous chapter. The prototype implements a web service provided by the latest version of the Apache-Tomcat web server. It represents the middle-tier in the three-tier architecture that can be integrated with various other commercial products. The Microsoft Access Database simulates the third-tier, while the first-tier can be any of the commonly used Web browsers. The prototype is designed to process everything in the server side and to dynamically create every response to the user.

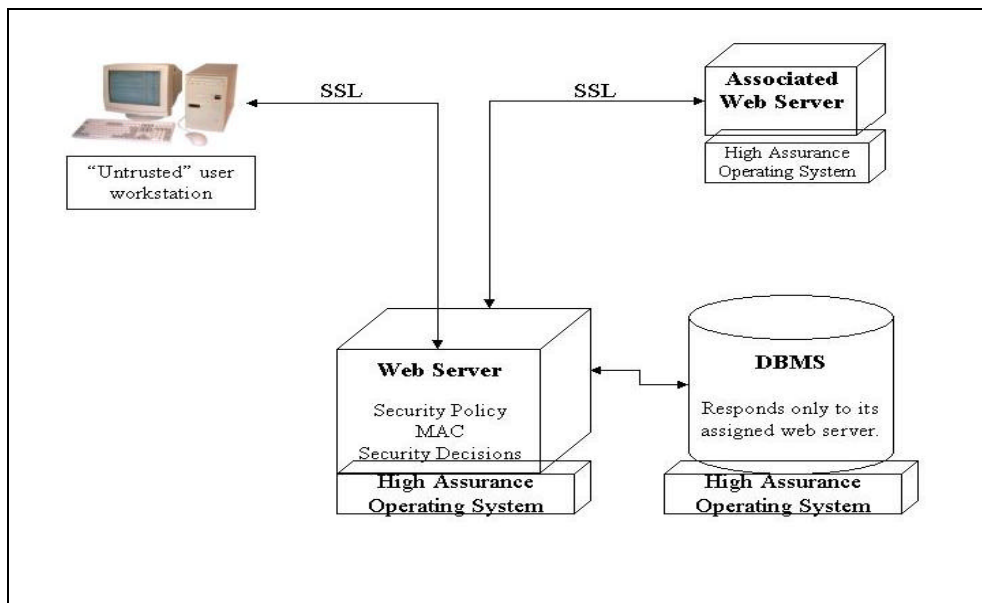


Figure 9. Security Architecture

The database is used to store document data, document metadata and user security data and to provide the requested resources to the service, which is the only software component that makes the security decisions according to the policy. The DB is also relied upon to keep separate the different levels of data. For simplicity, the database uses the "document storage" technique that was analyzed in the previous chapter. From the security point of view, it is better to have the web and the database server running on

different machines but in the same physical location in order to reduce potential malicious actions during their communication. If for any reason the web server and the database server must be separated, besides the known technologies for secure communication (such as SSL), a modified cryptographic integrity lock can be implemented. Similarly to the previous discussed Cryptographic Integrity Lock Architecture, the database would be assumed as untrusted, and the server will pass all the received information through the trusted filter.

The selected components (e.g. operating system, programming language, web server, database, etc) are those commonly used nowadays. Most of them are also open source or free software, which was the only reason they have been selected and not for their security features. The organization implementing the prototype, depending on its specifications and its requirements must choose the appropriate high assurance operating system that would support the web and database servers. Furthermore, all of today's architectures for secure authentication and authorization (e.g. PKI, Kerberos, X509) as well as those for secure transmission (e.g. encryption, SSL) between the user and the server can be easily integrated and used because the Java programming language and the XML supports all of them.

Even though the user's workstation is assumed as an "untrusted" environment, it is highly recommended to use a high assurance operating system, which will protect the user and the overall security of the organization. First of all it is important to establish a "trusted path" communication between the user and his or her own machine and then a secure communication path between the client machine and the server. Furthermore, care should be taken to prevent high level data from being written into low level server-managed objects by mistake or malice. This can be possible when a user had a previous higher level session, and then logged in (the same user or another one) at a lower level session. Residual data from the higher level session might be available on the client during the lower level session creating an "object reuse" problem. It can be addressed by simple techniques, such as a diskless workstation or an automatic user's machine reboot operation after every log off from the system. It is also a potential field for further research and future work.

2. Security Policies

As it is already mentioned, all the security decisions are made by the server, which is the only software component that enforces the security policy of the organization. The security manager of the prototype implements a Mandatory Access Control (MAC) policy using security objects that are dynamically generated from XML metadata security tags. For example, even though the information related to a user or a file are retrieved from the database, the security objects and their permissions are generated in the server according to the enforced security policy. Those objects are related to the file's (or user's) metadata tags, and are kept "alive" only during the current user's session and if for any reason the session is invalidated the objects are destroyed. Moreover, there is no way for a user, to directly communicate with the back-end database and request any resources because the database server accepts requests only from the assigned web server. Depending on the specific products, their physical location and their configuration, an authentication between the web server and the database server can be done once or in every request.

The security policy of the prototype system allows the authenticated user to open or to save documents at the level logged on and below. However, every time a user saves a file to the system, a user's object and the date and time are attached to the file, making that file unique. The main purpose of this operation is to make a clear distinction from the various versions of the same file, saved in the system by the same or other users. Only specifically authorized users may change the classification of documents. Concerning the communication between two servers, when a user requests a file that is located in an associated server, there are two different approaches depending on where the final decision is made. In the first approach, the requesting server, after its successful authentication, sends the user's object and the associated server makes the decision if the requested file is releasable. In the second approach, after the successful authentication of the requesting server, the associated server sends all the file names and their security objects to the requesting server, which will decide which of them are releasable to the specific user. In the prototype implementation, the second approach is used because the intention is to share the documents and not the user's information. Furthermore it

provides a better flexibility because each server can enforce differently its own policy to its own users.

B. STRUCTURE OF THE PROTOTYPE DEVELOPMENT

1. Main Idea

The prototype is mainly based on the object-oriented fundamentals, which are perfectly supported by the Java programming language and the XML features. The user of the system and its security attributes are represented as an object. Every file selected by the user either for opening or for storing is parsed and an associated security metadata labels object is created. The system compares those two objects and according to the security policy, determines the response to the user's request. The files' related objects, the user's objects and the processing software are all kept and are all managed in the server area, which is assumed to be a secure environment. The interface with the user is achieved though web pages that are generated dynamically according to every situation.

As already mentioned, the XML file's security attributes are mapped to a Java object. Due to the structure of XML, it is also possible to map every paragraph of the document to a Java object and to manipulate each paragraph differently. Even though this technique can be implemented easily from a technical point of view, there would be problems with the logic of the document itself. Accessing the paragraphs of a document that are not allowed due to their security attributes could make the document non-contiguous and potentially difficult to understand. The only exception is when the author of the document has the special training required to write those multi-level documents, which must be understandable, persistent and of course without revealing sensitive information by inference. This is an issue that the Intelligence Community Metadata Working Group has already noted in their reports and conferences.

2. Description of System's Logic—Flow Diagrams

The first page presented to the user by the server is a simple html file, which briefly describes the application and the necessary links to advance to the other pages. One of the links is "Help," which opens another html file with general directions about the use of the application and some answers on usual problems one might encounter. Another link, in the main page is "User Login," which opens a simple html file with a form waiting to receive the user's login name and password. The inputs of the login page

are sent to the first Java servlet, called *Login.class*, which has the responsibility to authenticate the user according to the user name and password received.

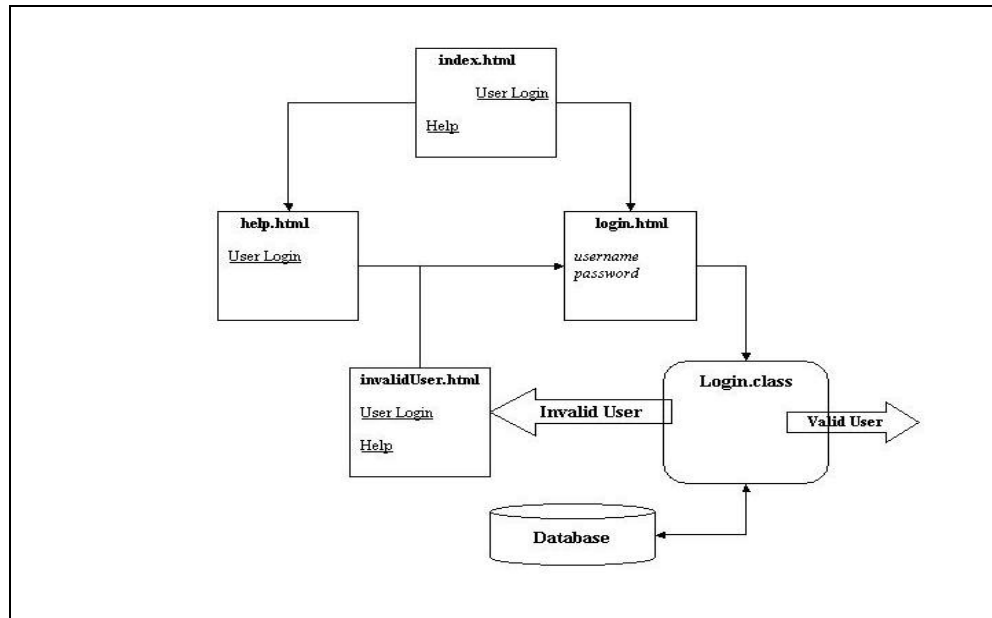


Figure 10. Login Flow Diagram

The *Login.class* initiates itself and attempts a connection with the third—tier database server. When the connection is established, this class queries the database using the user's credentials. The response from the database is processed inside this class. In case those credentials are not valid, an error HTML page is displayed with a respective message and the appropriate options to continue. If the user's credentials are valid according to the database, a number of queries are sent again to the database server to retrieve the authorizations of the user and those files are then available according to those authorizations. A *UserBean* object is created and attached to the new session. This object will be used during the whole procedure.

After the authentication and authorization of the user, the control is passed to the *UserOptions.class* that generates a web page with some of the user's personal data, such as last name, first name, and the available options. Those options are the main menu and they are presented in a drop down list. When one of them is selected, the control is passed to the *UserSelection.class*. The *UserSelection.class* does not generate any web

pages. It is only responsible for redirecting the control to the next respective servlet, according to the user's selection.

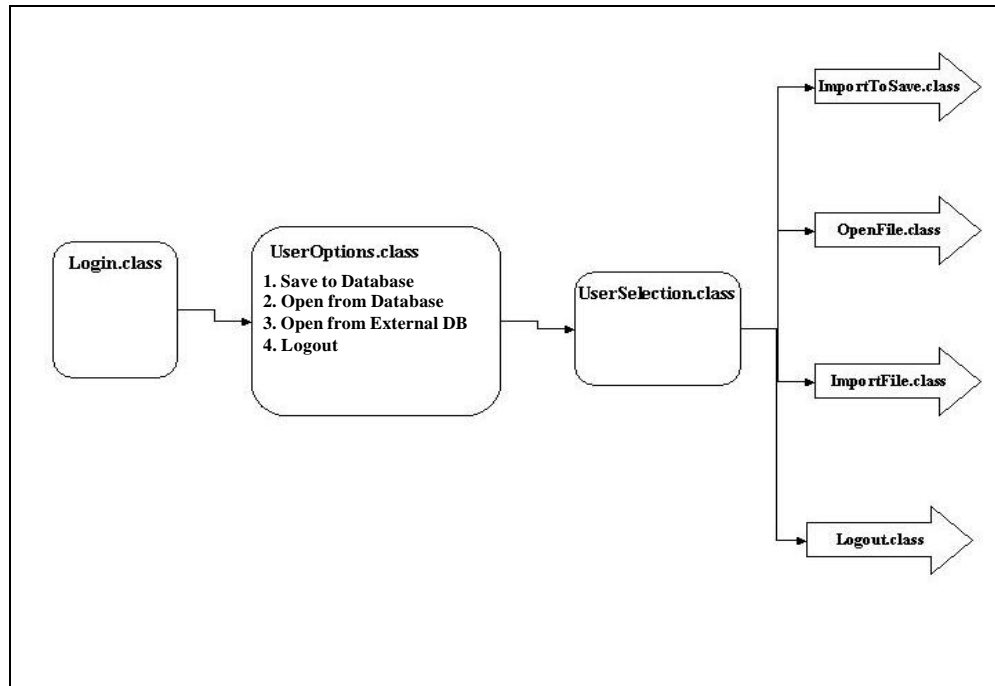


Figure 11. Main Menu Flow Diagram

As can be seen in Figure 11, the user is presented with four choices. The first choice, named “Save to Database,” is used when the user has created or copied a file inside his own machine and wants to send it to the server. The first servlet called in that option is *ImportToSave.class*, which generates a web page waiting for a filename to be sent to the server. The selection of the filename can also be done through the “browse” button that will initiate the “open file” window, provided by the operating system. Choosing from the local file system will be more convenient than writing the name of the file. On this web page, some of the user's data and some other administrative information that will be appended on the selected file are presented. The user cannot change any of his or her data associated with the file, because this is done on the server side, and their presentation on the page cannot affect their processing on the server, because the system does not accept those kind of information from the user. The user can only change the presented administrative information in order to characterize the file. Furthermore, the

user can select the classification of the file, which according to the enforced policy (prohibits “write-up” or “blind-write”) could not be higher than the user’s present classification. The presented classification choices are only those accepted from the logic of the system (e.g. “write-down”). Finally, the page displays the date and the time that will be appended to the file in order to be used later as the key reference for the version of the file.

When the user inputs the filename and selects the “upload to the server” button, the *SaveFile.class* is called. The first task of this class is to initiate the upload procedure. During the uploading, a temporary file position is created in the web server area in order to accommodate the file. After a successful uploading, the server parses the file in order to complete a number of required checks. The first of them is to identify that the file is an XML file and that it is well formed and valid according to the Intelligence Community Metadata Standard’s for Publications. This task is achieved by the *Echo24.class* that parses the file using one of the “SAX Parsers” provided by the “javax.xml” packet of Java. If the file is well formed and valid, a *FileTags.class* object containing all of its security attributes is created for that specific file. The system retains the *FileTags* object only during the required time for processing inside the server or until the end of user’s session.

During the uploading and parsing process, a number of messages are created and presented to the user in order to give him or her a view of the procedure. If the system encounters any problem or errors, during the above process, an appropriate message, indicating the cause is generated and displayed. If the process is completed successfully, the newly created *FileTags* object for that specific file as well as the *UserBean* object are sent to the *SecMetadataManager.class*. This class acts as the security manager. The important task that the *SecMetadataManager.class* performs is to compare the two objects and to generate the decision regarding that specific file and that specific user. When the user requests to store the file with a different classification than the XML security attributes contained in the file itself, an appropriate message is generated. This operation helps to prevent unintentional change of the file’s classification, made by user’s mistake or malicious code, to downgrade a document for disclosure to lower classification users. For example, when a user currently logged on as “SECRET,”

requests to save a file as “UNCLASSIFIED,” but the XML file contains security classification “CLASSIFIED,” then the system generates an “downgrade” message to warn of the difference. If the user confirms the change of the file from “CLASSIFIED” to “UNCLASSIFIED,” then the system downgrades the file and saves it in the respective area with the changed security attributes. According to the policy, there is no case for a user to have the capability to request the system to save a file with a classification higher than the classification for which the user is currently logged on. The *SecMetadataManager.class* is the one and only security enforcement module, which is always invoked for security decisions when a file is saved to the system.

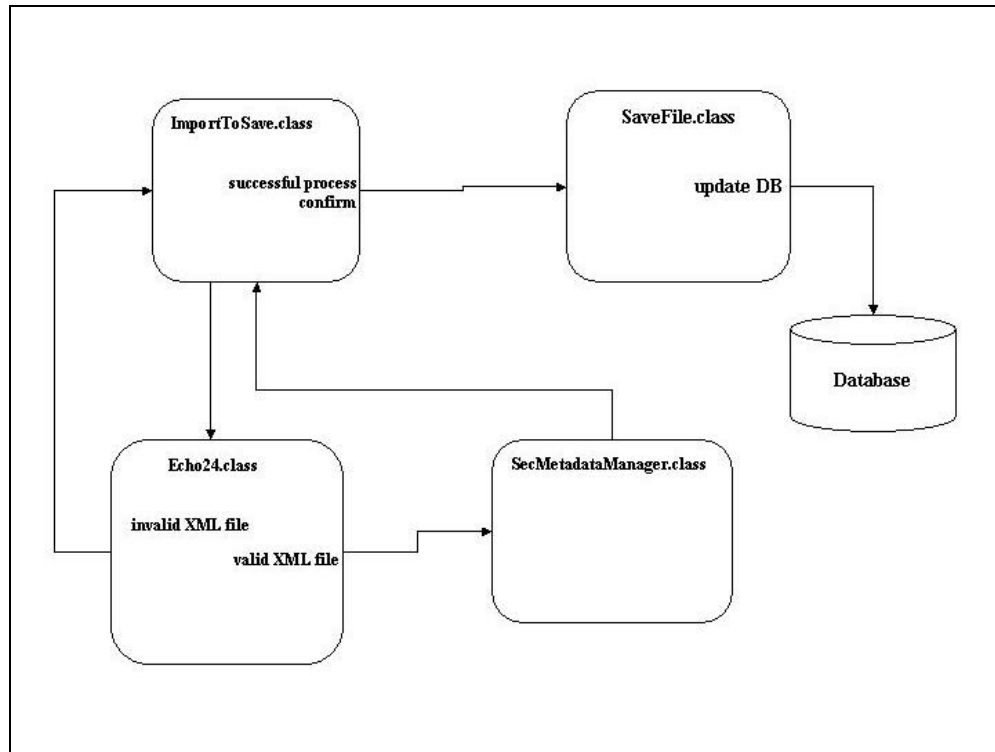


Figure 12. Save to Database Flow Diagram

Viewing the menu, the user may select the second choice named “Open from Database.” It is used when the user wants to retrieve a file from those stored in the system’s database. The first class is called *OpenFile.class*. This class opens a connection with the database and retrieves all the files that are available and can be released to the specific user according to his or her security attributes. User can also input filename

criteria so system can search among the releasable files. This operation is performed by generating queries to the database, according to user's security attributes and the enforced security policy. The *OpenFile.class* is the security enforcement module, which is always invoked for security decisions when a file is retrieved from the system. The name and the version (simulated by the date and the time the file entered the database) are presented to the user who must select one of the files to open. In the case that a user has a high classification, the files are separated and displayed in categories according to their classification levels.

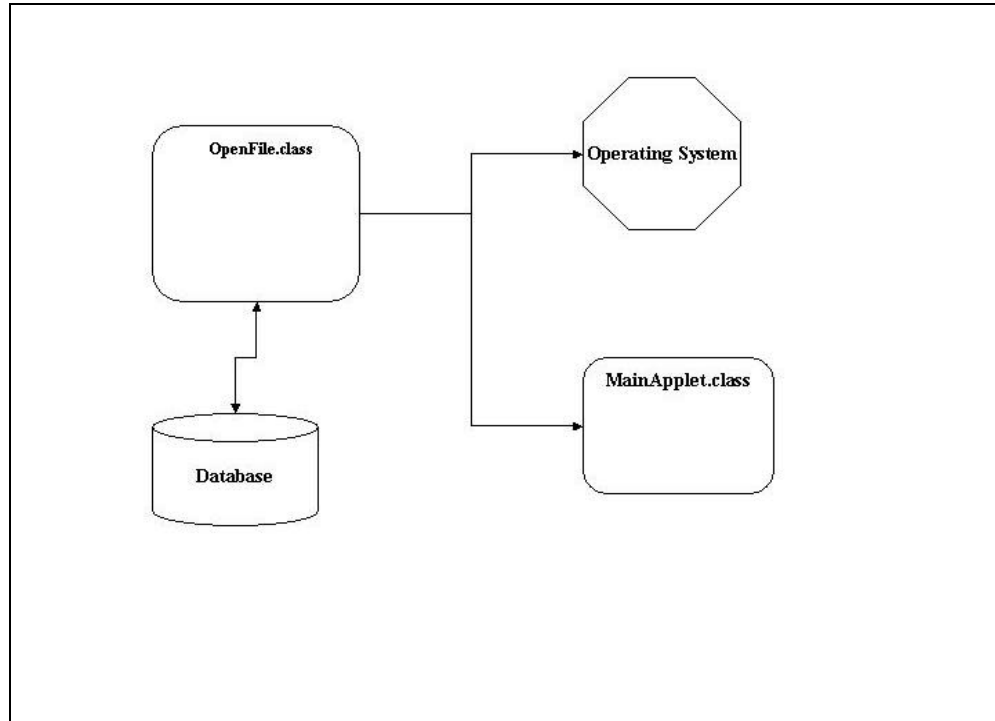


Figure 13. Open File Flow Diagram

The user selects a file and its respective filename is sent to the same *OpenFile.class*, which retrieves the file from the database, creates a *fileTag.class* object for the file, and then gives the user two opening options. The first one is to send the file to the user's workstation to directly open the file, where the underlying operating system will activate the default XML editor or where it will save the incoming file to a local directory. As it is mentioned in the beginning of this chapter, the user's operating system

is assumed as untrusted. The other opening choice is to send the file to the user's workstation to be processed by an applet simulating a distributed XML editor residing in the server. The first choice gives the user more flexibility but requires a powerful XML Editor installed in the user's machine and a well maintained, regarding the security, operating system. On the other hand, the applet-based XML Editor might restricts the user's capabilities but could also provide better security features due to the reduced user interaction. Furthermore, in an applet-based editor, many customized options, specific to the organization can be added or removed at the same time for all the users.

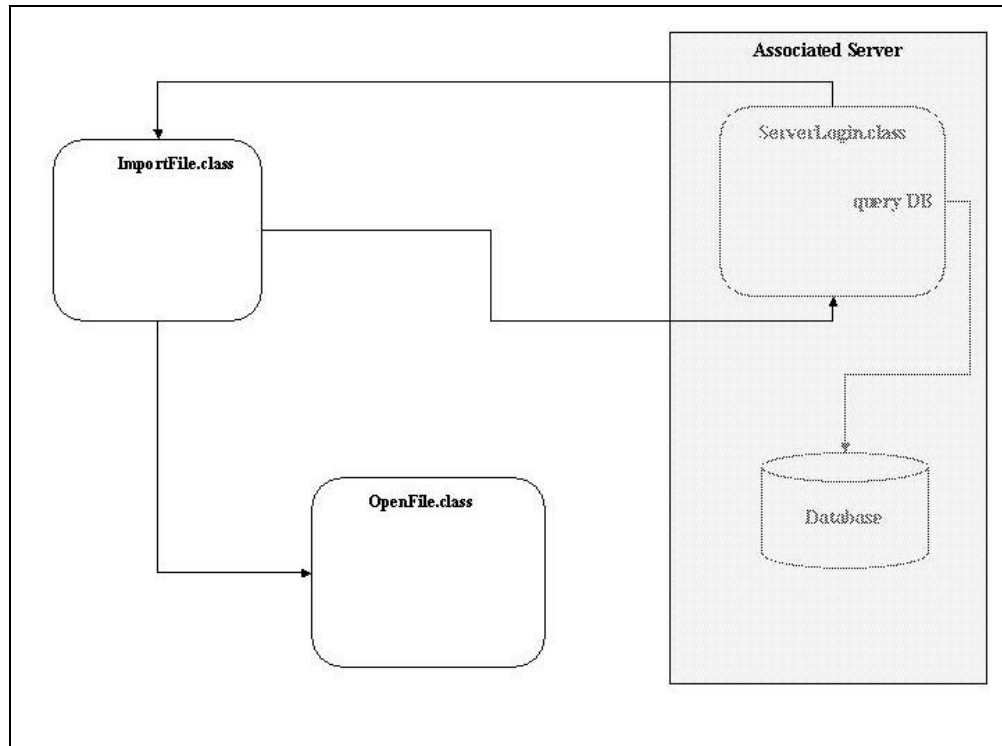


Figure 14. Open from External Database Flow Diagram

Returning again to the user's main menu, the third choice, "Open from External Database," is used when the user wants to obtain a file that is kept and controlled by another server. It is assumed that the user's server is already associated with other servers, running the same or similar implementation, and they have agreed to share their XML documents. Obviously, the number and the kind of the associated servers, depends

on the policy the organization currently uses and of course can be modified at any time restricting or allowing access.

If the user selects the third option, the first class called is *ImportFile.class*, and this in turn generates a web page on which the user must select the affiliated enclave or organization whose server is associated with the system. There is also a field where the user can input the complete filename or any portion of it so that the system can search among the releasable files. When the submit button is selected, the *ImportFile.class* initiates a connection with the respective server by calling the *Login.class* of the associated server. After a successful connection, an authentication and authorization occurs, and when they are completed successfully the releasable filenames and their security classifications are returned. The user's server has the responsibility to determine which files the specific user is allowed to obtain. Another possible approach could be to send the user's object to the associated server which will be responsible for making the decision. This approach is not recommended because it could reveal many information (directly or indirectly) about the users related to a server. The intention is to share only the documents and not the users' information.

The *ImportFile.class* compares the classifications of each file with the user's classification and keeps only those files that are releasable to the user. If the user has inserted a filename or any word previously, the system searches among the file names to find those that match. Finally, a web page is generated and the results are displayed through the *OpenFile.class*, which now assumes control and handles the file similarly to the previous choice.

The last option in the user's main menu is "Logout," which closes all the connections, deletes any temporary files and kills every object created during the user's session. The respective class is named *Logout.class*, and after a successful execution it generates a web page displaying a greeting message and links to the home page.

As a future work, many other options can be added to the main menu extending the performance of the implementation. One of those options could be the capability, for specifically authorized users, to save a document in an external database.

3. Analysis of the Java Servlets

The prototype is implemented using Java servlets running on the latest version of Apache—Tomcat. Their objectives, an analysis of their functionality, and their main methods are presented below

a. *Login.class*

The main objective of the *Login.class* servlet is to authenticate the user and to retrieve his or her authorizations. *Login.class* is the first servlet called after the user inputs his or her credentials to an HTML form. The *Login.class* begins by initializing the *ServletConfig.class*, which is its super class and thus inherits many properties. The next step is to open the connection with the third-tier database calling the *JdbcOdbcDriver.class* provided by the Sun Microsystems. The connection object created is attached to the session. The *Login.class* can accept either a GET or a POST request from the html file.

The first method called is *validateUser*, which makes the query to the database to determine if the username and password entered are contained in the database. When the database determines that the user's credentials are valid, a number of other methods are called in order to retrieve all the user's information. The user's personal and security information and any authorized files are all packed in a *UserBean.class* object. Then the *startSession* method is called. This instantiates a *RequestDispatcher* object, initiates a new Session object by attaching the newly created *UserBean* object, and transfers the control to the next servlet. In case of any error or if any exception is thrown, a respective error message is passed to the *exitPoint* method. This generates a dynamic HTML page displaying the cause of the problem.

b. *UserOptions.class*

After successfully authenticating and authorizing the user, the *Login.class* passes the control to the *UserOptions.class*, which creates a page presenting the user's information and the available options. The first method called is *printData* that obtains the personal and security data from the *UserBean* object and displays them to the user. The next method called is *printOptions* that creates a drop-down menu displaying the available options to the user. There is also a link in case the user wants to return to the previous page.

c. *UserSelection.class*

The user's selection is submitted via a POST request to the next servlet, the *UserSelection.class*. This servlet's only responsibility is to obtain the selected option and redirect the control to the appropriate servlet. The *doPost* method, using a number of "if" statements, determines what the selected option is and then calls the *startSession* method that dispatches a *RequestDispatcher* object to the respective servlet.

d. *ImportToSave.class*

The objective of the *ImportToSave.class* is to create the necessary page so that the user can input some required information about the file and, of course, the file's name. The first method called is the *printStandardMetadata* that draws a table and displays the user's data that will be appended to the file. The user cannot change those data. However, by calling the *dropDownClass* method, the user can change the file's classification, providing that the user, according to the policy, has an equal or higher classification. The *dropDownClass* method displays and permits him or her to choose only those allowed. In a similar way, the *dropDownCountry* method allows the user to select the countries to which this file will be releasable. Adding more methods like the *dropDownClass* or the *dropDownCountry*, an implementing organization can allow its users to select more options according to its policy.

After the *printStandardMetadata* method, the control is transferred to the *inputFile* method that creates the text field used for inserting the name of the file. This field is created by the "INPUT TYPE='file' " HTML tag that also provides a browse capability to the user. The necessary submit button, labeled "Upload to the server" is also displayed as well as a "Reset" button. Finally, the *exitLinks* method is called displaying the links to cancel the session and to login as another user or to go back to the main menu page and select another option.

e. *SaveFile.class*

The name of the file selected by the user is passed to the next servlet, *SaveFile.class* that is responsible for retrieving the file from the user and after a few checks to save it in the server. The method *uploadFile* is called within a "try-catch" block in order to be ready to catch any I/O exception that might be thrown by the process. A temporary directory and a temporary file are created in a specific server area to

accommodate the uploaded file. Due to the selected way of uploading the file (through the “ *INPUT TYPE='file'* ” HTML tag), the rest of the passed information accompanying the file are in the beginning of the input stream. The necessary processing will distinguish and separate those parameters from the file itself. Moreover, at this point, the system will determine if the name of the file is a valid XML filename, and if the file is starting with the XML version declaration. After completing those checks successfully, the reading-writing procedure starts.

After the whole file has been saved to the temporary server area without any errors or exceptions, the *checkingFile* method is called. Using the temporary name of the file, the *checkingFile* method creates an object of the *Echo24.class*, which will parse the XML file. The method *parseFile* of the *Echo24.class* is called obtaining the name of the file and the parsing procedure starts. The parser used by the *Echo24.class* is a *SAXParser* from the *SAXParserFactory* class of the “*javax.xml.parsers*” package. While parsing the file, a *FileTags* object is created by the *parseFile* method and returned to the *checkingFile* method if no errors are encountered. Returning a “null” *FileTags* object means that the parser found validation errors in the document, so an appropriate error page is created.

When the validation-parsing process is completed without errors, the *FileTags* object as well as the *UserBean* object are passed to the *SecMetadataManager.class*. The main task of the *SecMetadataManager.class* is to compare the passed *FileTags* and *UserBean* objects and to return the decision. In this specific situation the file is checked to determine the classification in which it will be stored in the server. The decision can be “auto-downgrade”, “downgrade,” “upgrade” or “normal.” “Auto-downgrade ” means that the file was characterized by the user with a higher classification than the specific user is permitted. So the system will automatically downgrade the classification of the file due to the user’s current logged on level. This feature enforces the policy that prohibits users to write in a higher level than the level they are currently logged on and could not be used as a normal procedure. If a user wants to “downgrade” a file that is already saved in the system with a high classification, he must log on, at least in the same level as the file. The decision “Upgrade” means that the user has the authority and asks to save the file with a classification higher than the file

already contained in its security attributes. It is actually only an observation of the system, which will not have any negative impact on the overall security of the system. The final possible decision is “normal,” which means that the security attributes are in agreement.

During the above procedure, the system generates appropriate messages to inform the user about the progress. Assuming that all the procedure are completed without any errors, the system displays the decision and waits for the user’s confirmation. That confirmation will trigger the transfer of the control to the next servlet.

f. UpdateDb.class

The next servlet called is *UpdateDb.class*, which as its name implies, will mainly update the database. First, the *copyFile* method is called in order to simulate the transfer and storage of the file to the database server and to an isolated secure area of the system. The next method called is the *updateDatabase* that updates the tables of the database containing the metadata of the file.

g. OpenFile.class

When the user selects the option “Open File” in the main menu, the *OpenFile.class* servlet is called. The main task of this servlet is to coordinate the opening procedure either for a file saved in the local or in an associated server. The *printAvaFiles* method displays all the files that are available to the specific user. Those files are retrieved from the database through the *findFiles* method, which opens a connection with the database and queries the respective table containing the file’s metadata. Similar to the *printAvaFiles* method is the *printServerAvaFiles* method, which is used when the files are obtained from one of the associated servers. The available files are displayed with their name and their version, namely, the date and the time saved in the system. The name of each file is presented as a link. When the user selects a file, the same *OpenFile.class* is reactivated but with a different sequence of methods, which are, determined from conditional “if” statements.

The system provides two different options to the user to open the file. The first choice is by using the operating system’s incoming file options, which in the case of Microsoft Windows can be to open the file by the default XML editor or to save it to the user’s machine for future process. The default application that Windows uses to open a

file can be changed at the “Folder Options” contained in the “Tools” menu in the Windows Explorer. The tab “File Types” will display the list of the file extensions and the applications to which each of them are associated. When the user has not defined an application for a specific type of file, the Internet Explorer will be used.

Another choice that the system provides is an applet-based XML editor that can be loaded on the user’s machine at the time a file is requested. This capability simulates a distributed XML editor, provided by the server in the case where the user does not have one installed locally. It is implemented by calling the *MainApplet.class*, which is a Java applet extending the functionality of the *Japplet.class* contained in the *java.swing* package. The name of the file to be opened is passed to the *MainApplet.class* applet using the “PARAM NAME=*filename* VALUE=*TheActualFilename*” html tag.

h. ImportFile.class

The *ImportFile.class* objective is to initiate the import procedure from one of the associated servers of the system. Its basic method is the *inputUrl* that creates one “radio” button for each of the affiliated servers and a text field for the name of the requested file. The system accepts an empty filename and translates as “find them all.” The necessary submit and reset buttons are also created and displayed to redirect the control to the next servlet and to clear the field, respectively.

i. FindUrl.class

The requested filename and the associated server where the file resides are passed to the *FindUrl.class*. This will establish a connection with the server and then retrieve the file. Using multiple “if-else” statements, the *FindUrl.class* determines the progress of the procedure. In the initial stage, the *getConnected* method is called with the name of the server as an argument. The *getConnected* method using a “try-catch” block encodes the URL of the selected server and calls the *ServerLogin.class*. The *ServerLogin.class*, as the name implies, is a login class dedicated to the authentication and authorization of the calling server. Its logic is similar to the *Login.class* mentioned above, which authenticates and authorizes a user. After an established connection and a successful authentication, the *ServerLogin.class* of the associated server retrieves the available files and transmits their metadata back to the calling server.

The *FindUrl.class* receives the available files from the associated server, and assuming that a filename (or a portion of it) has been entered, disregards those files whose names do not match. From the remainder, those that do have matching filename, the system compares the user's security attributes to those of each file. If a file has a higher security classification than the user possesses at this session, then the file is removed from the list. Finally, the remaining list of files, if any, is passed to the *OpenFile.class*, which will proceed to the opening procedure as already described. The only difference now is that instead of the *printAvaFiles* method, the *printServerAvaFiles* method will be called.

j. *Logout.class*

The *Logout.class* purpose is to terminate the user's session. Its basic method is the *processRequest* that ends the session by calling the *invalidate* method of the *HttpSession* class of the *javax.servlet.http* package.

THIS PAGE INTENTIONALLY LEFT BLANK

V. EXPERIMENTATION

A. DEMONSTRATION SCENARIOS

1. Introduction

The experimentation was conducted on the prototype implementation to determine its usability and performance characteristics. The potential users of the prototype can be either “internal” or “external” to a given server. Internal is when the user is locally connected to the server. An external user is not locally connected to the server, but is locally connected to a remote server. The demonstration scenarios presented below are intended to mimic the actual process anticipated in a large organization or in many associated medium organizations.

2. Scenario 1 – Internal User Stores a Document to the System

The objective of this scenario is to demonstrate the process required when an internal user authors an XML document and wants to store it in the system, making it available to other users. It is assumed that the user is within the organization, and the file is saved inside the local machine (either on the hard drive or in any other means accessible from the local file system).

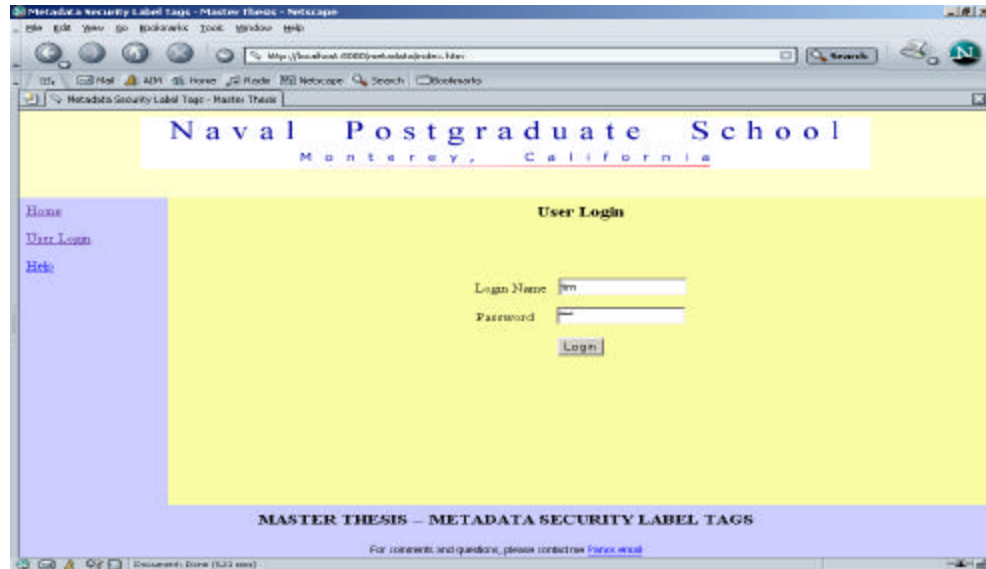


Figure 15. Login Screen

The file could have been created in the local machine using an XML Editor, or downloaded or copied from a disk. Finally, it is assumed that the user has a browser installed and has already registered in the system and has acquired a valid username and password and thus can be authenticated by the system. The screen shots shown in this scenario employed the Netscape Navigator.

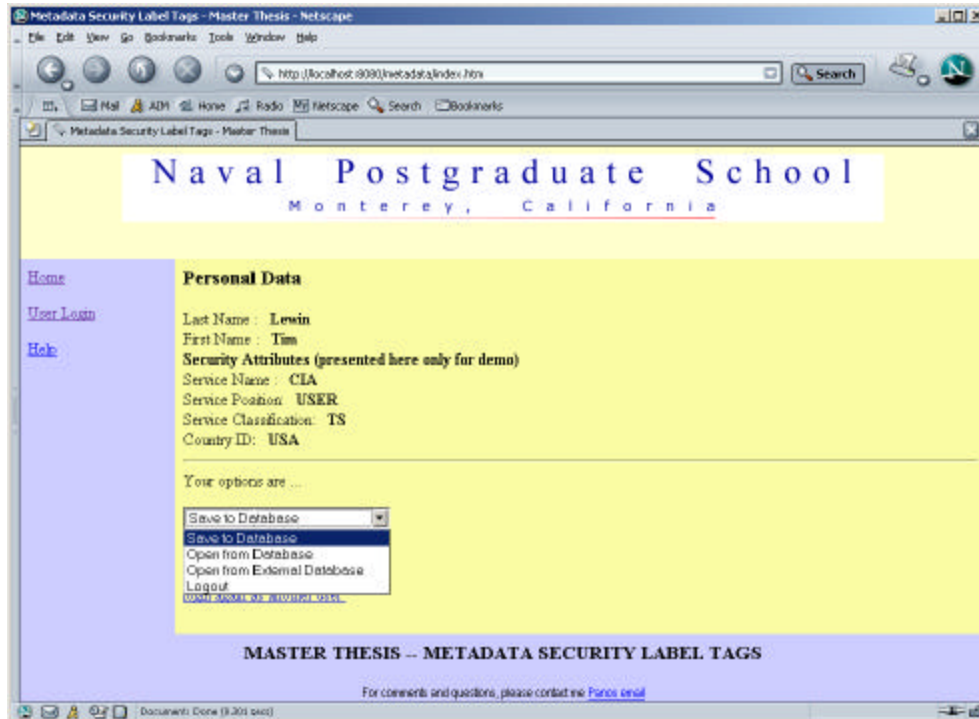


Figure 16. First Option in the Main Menu

The authenticated user selects the first option (Save to Database) in the main menu after the successful authentication and authorization of the system. The “Save to Database” screen is generated waiting for the user’s selections.

Selecting the “Browse” button, access to the local file system is provided to the user in case the exact file position is not known. The allowed file classifications presented to the user cannot be higher than the user’s classification.

The system generates messages showing the progress and requests the user's confirmation to proceed to the final step of updating the database if there are not any errors or exceptions.

3. Scenario 2 – Internal User Retrieves a Document from the System

This scenario demonstrates the ability of the system to provide an authorized user with one of the files stored in the database. It is assumed that the user is within the organization and has installed an XML editor or the Java Virtual Machine (VM) on his or her own machine. Moreover, it is assumed that the user has customized the local operating system to initiate the installed XML Editor when an XML file is received. The screen shots shown in this scenario were obtained from Microsoft Internet Explorer 6.0.

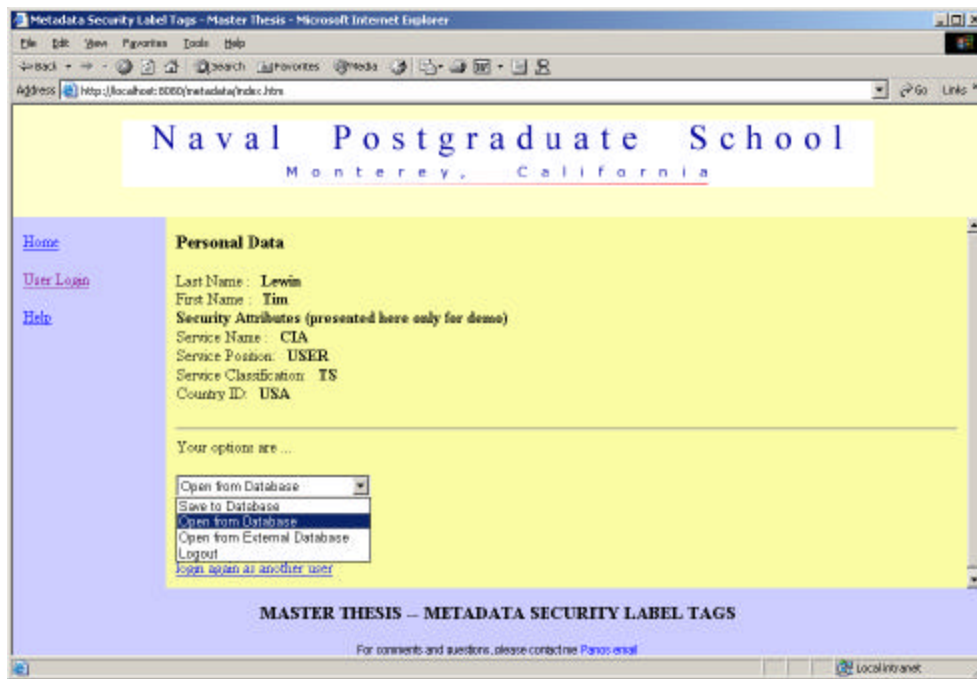


Figure 19. Second Option in the Main Menu

When the second option from the main menu is selected, the available files to the user are displayed. They are categorized depending on their classification, providing that the user has a high enough classification to allow him or her to view many layers.

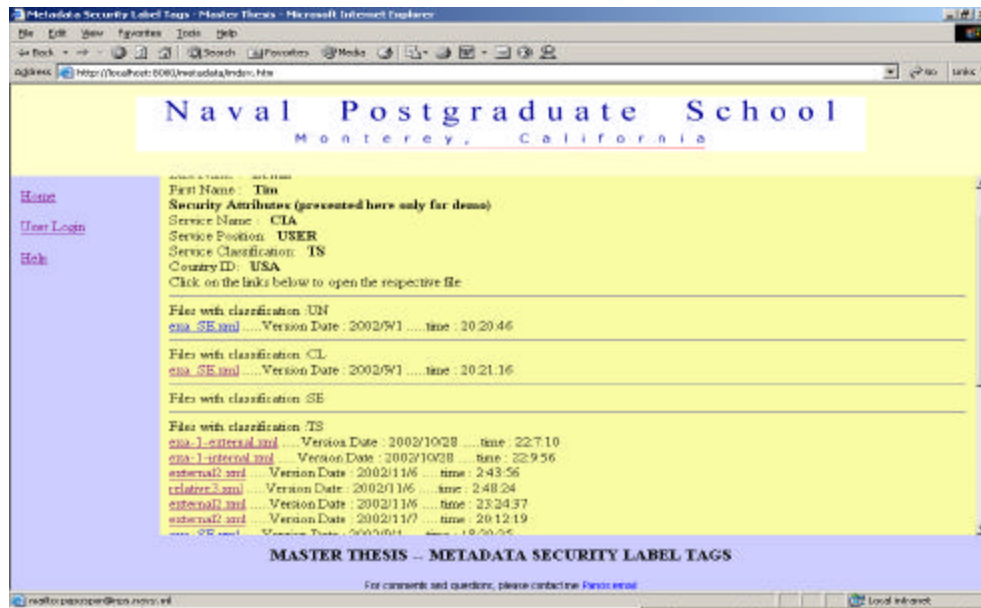


Figure 20. Available Files to Open

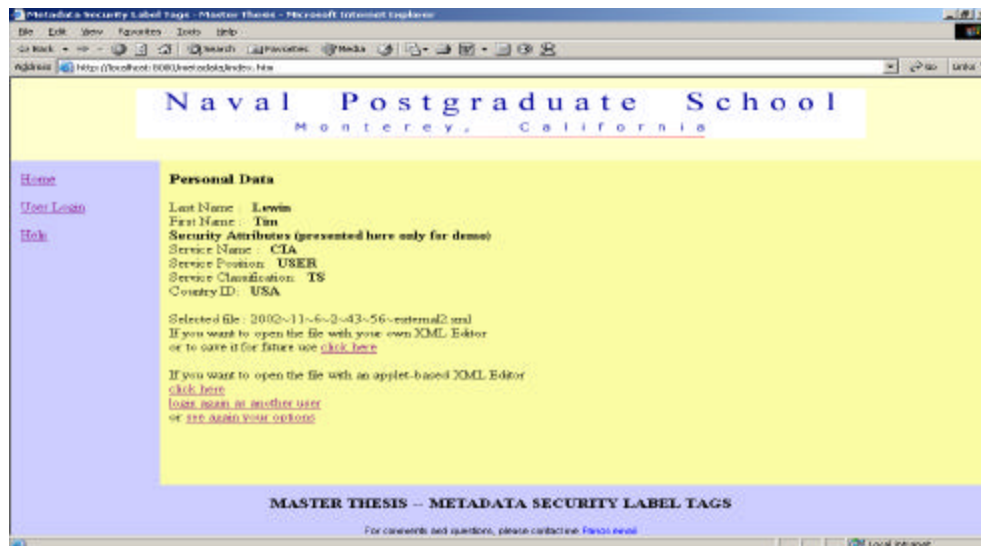


Figure 21. Choose an Editor Screen

Each file is a link, which, when selected initiates the next page presenting the two options of how that file can be opened. If the user chooses the option of the installed XML Editor, the operating system presents the available options, which either opens the file using the default Editor or saves the file locally for future use.

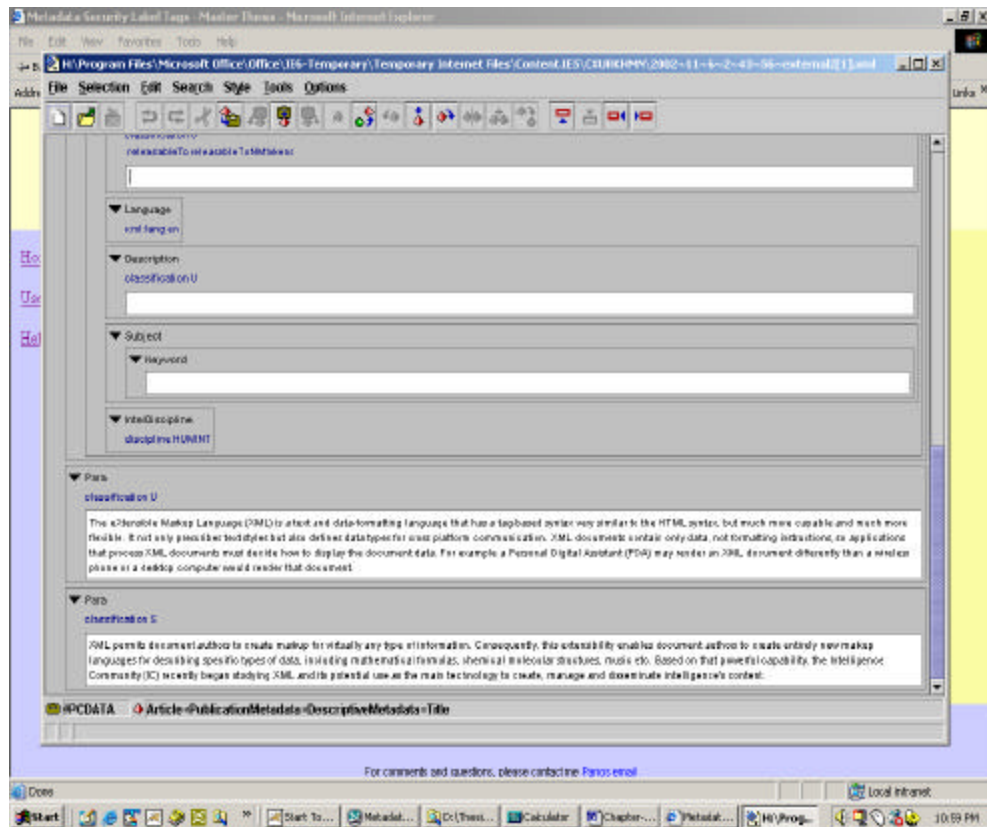


Figure 22. XMLMind Editor Opened the File

If the user chooses to open the file using the applet-based XML Editor and providing that the Java VM is installed, the applet is loaded and initiated and presents the selected file.

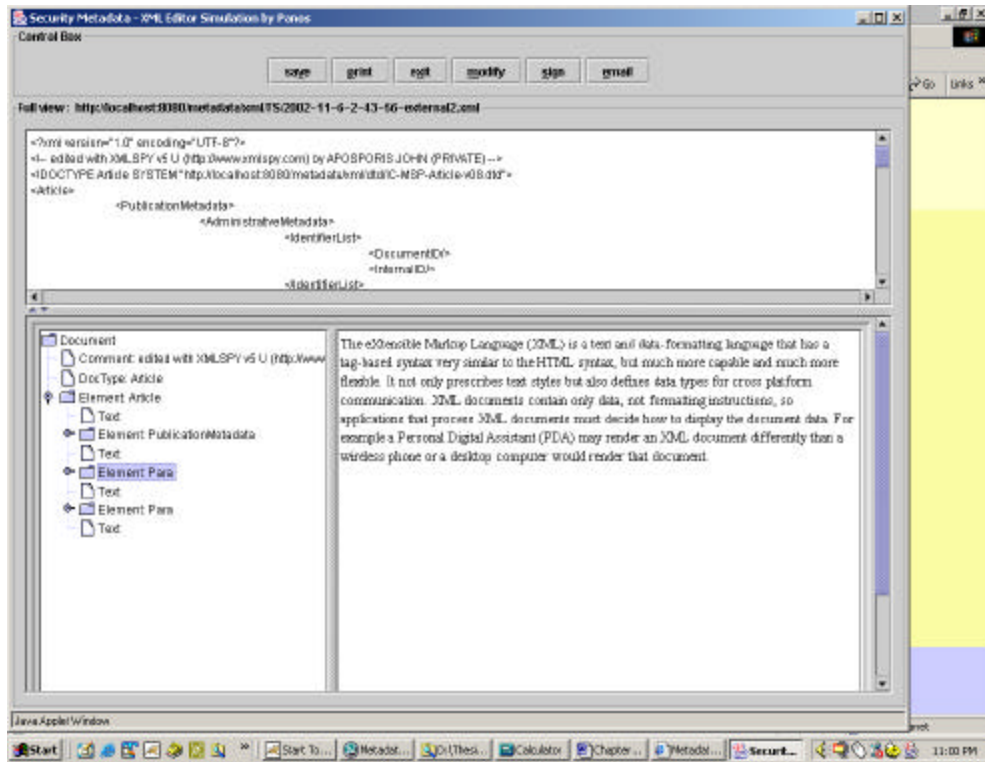


Figure 23. Applet Based Editor Opened the File

4. Scenario 3 – External User Retrieves a Document from the System

This scenario demonstrates the ability of the system to provide one of the files stored in the database to an authorized external user. It is assumed that the user belongs to one of the associated organizations or enclaves that share their documents. After the desired file is retrieved from the associated server, the procedure is similar to the previous. Thus an XML editor or the Java Virtual Machine (VM) must be installed on the user's local machine.

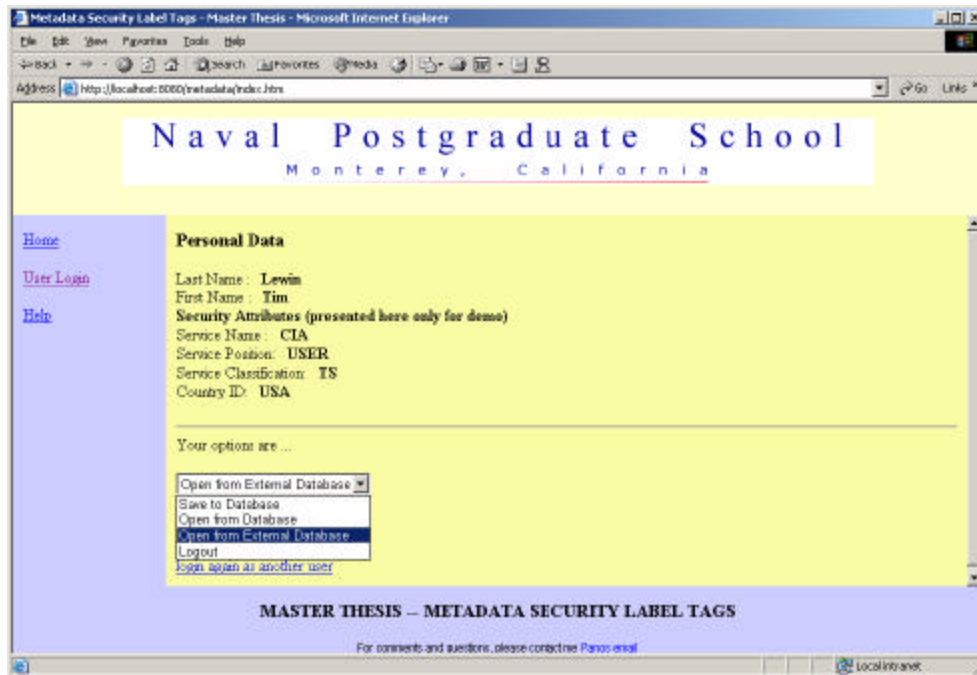


Figure 24. Third Option in the Main Menu

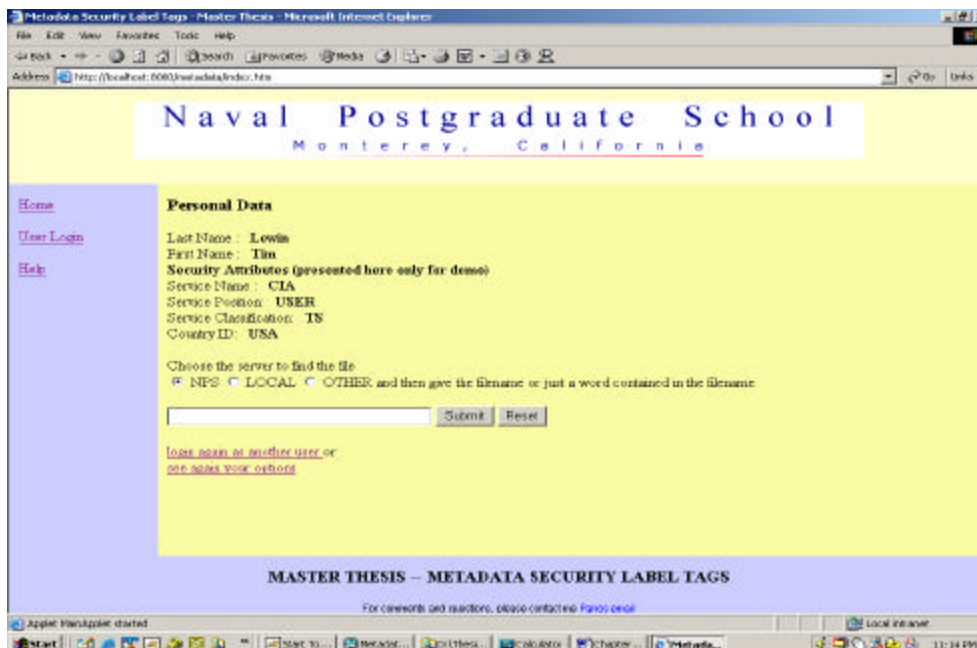


Figure 25. Choose an Associated Server Screen

The user selects the server, in which the file resides as well as the name of the file or at least the beginning of it. The system attempts a connection with the selected server as soon as the “Submit” button is pressed. Upon a successful connection between the two servers, an authentication and authorization process is started. When completed without errors, the available filenames are retrieved and sent back to the requested server for further processing. If that processing results in a file that matches, the inserted filename can be released to that specific user and a new request is sent again to the associated server to pass the file itself. In case the processing results in more than one file, then all of them are presented to the user to select.

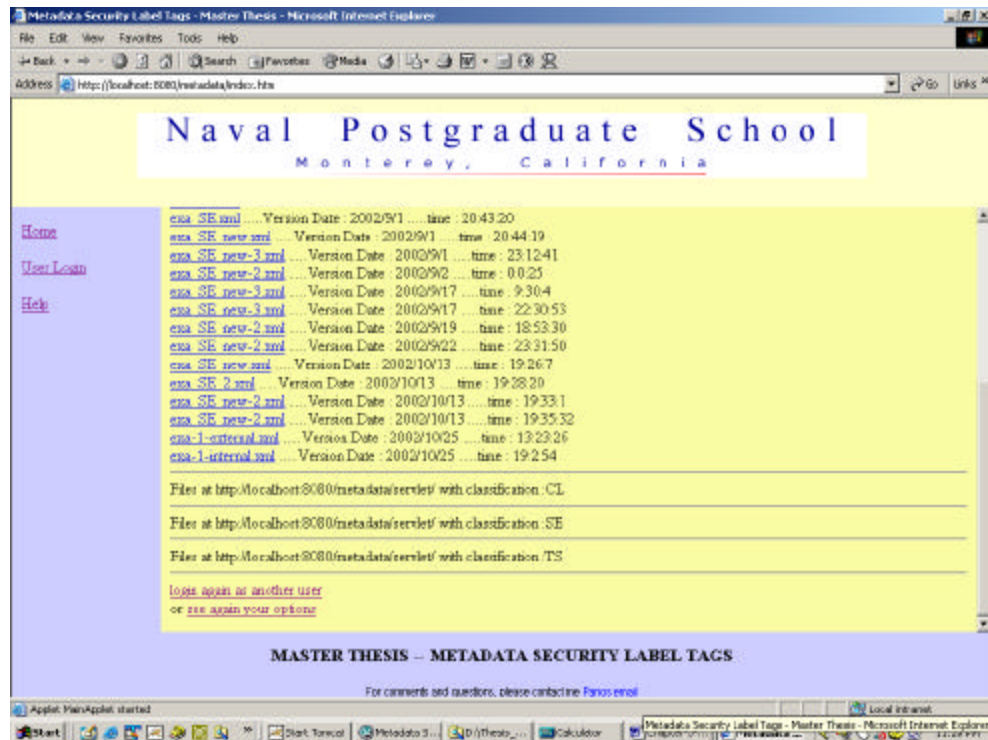


Figure 26. Available Files on an Associated Server

From the point where the file itself is retrieved from the associated server, the procedure is exactly the same as the opening procedure when the file is saved locally.

B. SUBJECTIVE PERFORMANCE EVALUATION

The prototype development and implementation is based on the Microsoft Windows 2000 operating system, the Java 2 Enterprise Edition integrated platform and the Apache-Tomcat web server. The Microsoft Access simulates the database management system. Various XML Editors such as “XMLSpy,” “XMLMind” and Web browsers, such as the Internet Explorer and Netscape Navigator are tested and used to develop and test the system. Java platform, Apache web server and Netscape are freeware and open-source products making them easily customized to the specific needs of every application. Almost all of them (except Microsoft Access) have adequate capabilities to support general requirements to manipulate, distribute and store XML documents.

However, because the system design extensively uses security markings that are represented as objects, and because the system itself bases its decisions on those objects, a high assurance operating system must be used in an operational implementation. The high assurance operating system must be used to accommodate at least the web server and the DBMS but it is also highly recommended for the user’s environment. When the performance requirements are similar to those used in the electronic business world the main concern is focused on the security of the underlying operating system, which supports the security of the entire implementation.

The Apache-Tomcat web server has a suitable overall performance that has made it one of the most popular and most reliable products in the highly demanding business world. Its open-source base and its efficiency in almost all the operating systems make it a good solution. Furthermore, because Apache-Tomcat is coded and based in Java, any application using the same programming language creates a better combination with even better performance. Java acquires a large number of capabilities and advantages that can expand and can improve the efficiency of the implemented application. Due to the object-oriented structure of Java, any new features or improvements can be easily attached in the future, keeping the entire solution always updated.

Regarding the specific tools an end user must use, such as web browsers or XML Editors, most of the commercial products tested were found to provide acceptable performance. However, there would be even better results if an integrated product could

perform both jobs (browsing the web and editing an XML document) with the same high level of efficiency and performance. This may be a worthwhile area for research and development in the future.

C. CONCLUSIONS - LESSONS LEARNED

XML is one of the most promising and rewarding technologies of today's demanding world. Especially when the environment or the platform is object oriented, XML technology can be easily integrated and help to increase overall performance. I have shown that by using XML and its metadata, developers cannot only manipulate documents fast and efficiently but can also do so securely and with flexibility and interoperability.

The use of XML inside a large organization can improve the performance and the efficiency in the production, distribution and storage of documents while sharing those documents among various enclaves. Furthermore, XML itself provides the capability to attach security attributes to granular elements. This capability makes XML much more advantageous than HTML. I have shown that many XML security concerns can be addressed by using current security technologies. Encryption, Digital Certificates, Digital Signatures are among those technologies that can assure the integrity and security of XML documents.

A highly secure operating system must be selected to accommodate and to support the main components of the implementation. Even though Microsoft virtually dominates the market and improves the security features of its products every day, a few other choices (Unix, Solaris) can achieve much better security. Starting from an operating system with advanced security capabilities, an implementing organization can select among the various existing frameworks. Almost all the major vendors of the market (Microsoft, Sun Microsystems, Oracle, etc.) have developed technologies that allow programmers and their organizations to use XML and its features. It is only a matter of the specific organizational requirements to determine which framework to use, due to the fact that all of the frameworks provide interoperability and extensibility through XML. Sun's Microsystems Java 2 Enterprise Edition and Microsoft's "dot " NET framework are the most powerful, robust and complete frameworks to support and implement an efficient and secure solution.

Finally, a great variety of products and applications, such as Web browsers, Web servers, XML Editors and Database management systems, can be used on top of the selected framework to support specific needs. Many of those products integrate broad capabilities and provide fairly complete solutions. The final selection of those products should be according to the specifications and requirements within the organization.

APPENDIX A. APPLICATION PROGRAMMING INTERFACE (API)

The Application Programming Interface (API) presented on the next pages is created by the “javadoc” utility, provided by the Sun Microsystems Java 2 Standard Edition.

Package dbsection

Class Summary	
<u>DBConnecti onBean</u>	This class implements a bean that is used for many other classes to process all transactions with the backend database.
<u>Echo24</u>	The class Echo24 is responsible for parsing the passed file and to extract all of its elements and attributes.
<u>FileTags</u>	The class FileTags is responsible for creating an object containing all the security elements and attributes of a specific file.
<u>FindUrl</u>	The FindUrl class defines a servlet that is responsible for connecting the system with the requested server, for retrieving the releasable files from that server and for deciding which of those files the user is allowed to access.
<u>ImportFile</u>	The ImportFile class defines a servlet that is responsible for creating a web page to help the user input the file and to help the server know where that file is located in order to open it.
<u>ImportToSave</u>	The ImportToSave class defines a servlet that is responsible for creating a web page to help the user input the file to be saved, as well as some information related to the file
<u>Login</u>	The class Login defines a servlet that is responsible for authenticating the users trying to login.
<u>Logout</u>	This class defines a servlet that logs out a user and ends his or her session.
<u>MetaTags</u>	The MetaTags class codes the tags of a file as an object.
<u>OpenFile</u>	The class OpenFile defines a servlet that is responsible for opening any file a user may select. Either the file is saved locally or in another server.
<u>SaveFile</u>	The SaveFile class is responsible for uploading the file from the user's machine to a temporary area of the system and to check and parse the file.
<u>SecMetadat aManager</u>	The class SecMetadataManager is responsible for comparing the received UserBean and FileTags objects and to make the necessary decision according to the policy.
<u>ServerLogin</u>	The class ServerLogin defines a servlet that is responsible for authenticating associated servers trying to login.
<u>UpdateDb</u>	The UpdateDb class receives an already parsed file, updates the database, and saves the file in the "secure" server area.

<u>UserBean</u>	This class defines a bean used to maintain the user's data during the session.
<u>UserOptions</u>	The class UserOptions creates a page displaying the user's information (personal and security) and the available options for that session.
<u>UserSelection</u>	The UserSelection class is responsible for getting the user's selection from the main menu and for redirecting the control to the respective servlet for further processing

Hierarchy for Package dbsection

Class Hierarchy

- class java.lang.Object
 - class dbsection.CalendarBean
 - class dbsection.CommonData
 - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
 - class java.awt.Container
 - class java.awt.Window (implements javax.accessibility.Accessible)
 - class java.awt.Frame (implements java.awt.MenuContainer)
 - class javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
 - class dbsection.TextSamplerDemo
 - class dbsection.CustomerBean
 - class dbsection.[DBConnectionBean](#)
 - class org.xml.sax.helpers.DefaultHandler (implements org.xml.sax.ContentHandler, org.xml.sax.DTDHandler, org.xml.sax.EntityResolver, org.xml.sax.ErrorHandler)
 - class dbsection.[Echo24](#)
 - class dbsection.[FileTags](#)
 - class javax.servlet.GenericServlet (implements java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig)
 - class javax.servlet.http.HttpServlet (implements java.io.Serializable)
 - class dbsection.CreateFile (implements java.io.Serializable)
 - class dbsection.[FindUrl](#) (implements java.io.Serializable)
 - class dbsection.[ImportFile](#)
 - class dbsection.[ImportToSave](#) (implements java.io.Serializable)
 - class dbsection.ListReservations
 - class dbsection.[Login](#)
 - class dbsection.[Logout](#)
 - class dbsection.[OpenFile](#)
 - class dbsection.[SaveFile](#) (implements java.io.Serializable)
 - class dbsection.[ServerLogin](#)
 - class dbsection.[UpdateDb](#)

- class dbsection.[UserOptions](#)
 - class dbsection.[UserSelection](#)
- class dbsection.[MetaTags](#)
- class dbsection.[SecMetadataManager](#)
- class dbsection.ServletUtilities
- class dbsection.[UserBean](#)

dbsection

Class DBConnectionBean

java.lang.Object

|

+dbsection.DBConnectionBean

public class **DBConnectionBean**

extends java.lang.Object

This class implements a bean that is used for many other classes to process all transactions with the backend database. It is a unique point of connection with the database, using the JDBC API.

Field Summary

private java.sql.Connection	con
--------------------------------	---------------------

private boolean	<u>connected</u>
private java.lang.String	<u>driver</u>
private boolean	<u>loaded</u>
private java.lang.String	<u>password</u>
private java.lang.String	<u>primaryKeyQuery</u>
private java.lang.String	<u>query</u>
private java.sql.Statement	<u>statement</u>
private java.lang.String	<u>update</u>
private java.lang.String	<u>url</u>
private java.lang.String	<u>user</u>

Constructor Summary

[DBConnectionBean](#) ()

Method Summary

void	<u>DBConnectionBean</u> () The default constructor of the class that sets the appropriate url and driver for the database
int	<u>getPrimaryKey</u> () Obtains a new primary key in the table specified by the current primary.
java.sql.ResultSet	<u>getQuery</u> () Queries the DB and return the result ResultSet object
int	<u>getUpdate</u> () Executes an update query.
boolean	<u>isClose</u> () This method closes the connection with the database
private boolean	<u>isConnected</u> ()

	Establish the connection with the DB.
boolean	<code>isLoaded()</code> Loads the driver
java.sql.ResultSet	<code>query()</code> (java.lang.String query) Sets the query string.
void	<code>setDriver()</code> (java.lang.String newDriver) Sets the driver for the database
void	<code>setPassword()</code> (java.lang.String newPassword) Sets the DB password.
void	<code>setPrimaryKeyQuery()</code> (java.lang.String newPrimaryKeyQuery) Sets the primary key for the query.
void	<code>setQuery()</code> (java.lang.String query) Sets the query.
void	<code>setUpdate()</code> (java.lang.String update) Sets the update query.
void	<code>setUrl()</code> (java.lang.String newUrl) Set the URL of the database.
void	<code>setUser()</code> (java.lang.String newUser) Sets the DB user name.
int	<code>update()</code> (java.lang.String updateQuery) Executes an update query by calling the <code>getUpdate</code> method

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Field Detail

loaded

private boolean **loaded**

connected

private boolean **connected**

url

private java.lang.String **url**

driver

private java.lang.String **driver**

user

private java.lang.String **user**

password

private java.lang.String **password**

con

private java.sql.Connection **con**

statement

private java.sql.Statement **statement**

query

private java.lang.String **query**

update

private java.lang.String **update**

primaryKeyQuery

private java.lang.String **primaryKeyQuery**

Constructor Detail

DBConnectionBean

public **DBConnectionBean()**

Method Detail

DBConnectionBean

public void **DBConnectionBean()**

The default constructor of the class that sets the appropriate url and driver for the database

Returns:

void.

setUrl

public void **setUrl**(java.lang.String newUrl)

Set the URL of the database.

Parameters:

`newUrl` - The new URL

Returns:

void.

setDriver

public void **setDriver**(java.lang.String newDriver)

Sets the driver for the database

Parameters:

`newDriver` - The new driver.

Returns:

void.

setUser

public void **setUser**(java.lang.String newUser)

Sets the DB user name.

Parameters:

`newUser` - The new user name.

Returns:

void.

setPassword

public void **setPassword**(java.lang.String newPassword)

Sets the DB password.

Parameters:

`newPassword` - The new password.

Returns:

void.

isLoading

public boolean **isLoading**()

Loads the driver

Returns:

True if the driver is loaded successfully

isConnected

private boolean **isConnected**()

Establish the connection with the DB.

Returns:

True if the connection is established.

query

public java.sql.ResultSet **query**(java.lang.String query)

Sets the query string.

Parameters:

query - The new query string.

Returns:

A ResultSet object containing the results of the query

getQuery

public java.sql.ResultSet **getQuery**()

Queries the DB and return the result ResultSet object

Returns:

A ResultSet object containing the results of the query

setQuery

public void **setQuery**(java.lang.String query)

Sets the query.

Parameters:

query - The new query.

Returns:

void.

setUpdate

public void **setUpdate**(java.lang.String update)

Sets the update query.

Parameters:

update - The update query.

Returns:

void.

update

public int **update**(java.lang.String updateQuery)

Executes an update query by calling the getUpdate method

Parameters:

updateQuery - the update query

Returns:

An integer representing the primary key or -1 if an error happens.

getUpdate

public int **getUpdate()**
Executes an update query.

Returns:

An integer representing the primary key or -1 if an error happens.

isClose

public boolean **isClose()**

This method closes the connection with the database

Returns:

A boolean true if the connection is closed successfully

setPrimaryKeyQuery

public void **setPrimaryKeyQuery**(java.lang.String newPrimaryKeyQuery)

Sets the primary key for the query.

Parameters:

newPrimaryKeyQuery - The new primary key for that query

Returns:

void.

getPrimaryKey

public int **getPrimaryKey()**

Obtains a new primary key in the table specified by the current primary. The result set of the ordered set of existing primary keys is examined sequentially, until the smallest non used positive integer is found. This method may only be used for tables using integers as a primary key.

Returns:

the new primary key or -1 if an error happens.

dbsection

Class Echo24

java.lang.Object

|

+org.xml.sax.helpers.DefaultHandler

|

+**-dbsection.Echo24**

All Implemented Interfaces:

org.xml.sax.ContentHandler, org.xml.sax.DTDHandler, org.xml.sax.EntityResolver,
org.xml.sax.ErrorHandler

public class **Echo24**

extends org.xml.sax.helpers.DefaultHandler

The class Echo24 is responsible to parse the passed file and extract all of its elements and attributes

Field Summary	
private java.io.File	<u>file</u>
private dbsection.FileTags	<u>fileTags</u>
private int	<u>indentLevel</u>
private java.lang.String	<u>indentString</u>
private int	<u>index</u>
private boolean	<u>nonusSecElementFound</u>
private static java.io.Writer	<u>out</u>
private java.lang.String[]	<u>parsedFile</u>
private java.lang.String	<u>secCategory</u>
private boolean	<u>secElementsFound</u>
private boolean	<u>usSecElementFound</u>

Constructor Summary	
<u>Echo24</u> ()	The default constructor of the class

Method Summary	
void	<u>characters</u> (char[] buf, int offset, int len)
private void	<u>emit</u> (java.lang.String s)
void	<u>endDocument</u> ()
void	<u>endElement</u> (java.lang.String namespaceURI, java.lang.String sName, java.lang.String qName)

<code>java.io.File</code>	<u>getFile()</u> Gets the file contained in the object
<code>java.lang.String[]</code>	<u>getParsedFile()</u> Gets the parsed file contained in the object
<code>dbsection.FileTags</code>	<u>mainNew()</u> (<code>java.lang.String argv</code>) This method parses the file in the passed URL by using one of the SAXParsers from the SAXParserFactory
<code>private void</code>	<u>nl()</u>
<code>dbsection.FileTags</code>	<u>parseFile()</u> (<code>java.lang.String filename</code>) This method parses the file by using one of the SAXParsers from the SAXParserFactory
<code>void</code>	<u>setDocumentLocator()</u> (<code>org.xml.sax Locator l</code>)
<code>void</code>	<u>startDocument()</u>
<code>void</code>	<u>startElement()</u> (<code>java.lang.String namespaceURI</code> , <code>java.lang.String lName</code> , <code>java.lang.String qName</code> , <code>org.xml.sax.Attributes attrs</code>)

Methods inherited from class `org.xml.sax.helpers.DefaultHandler`

`endPrefixMapping`, `error`, `fatalError`, `ignorableWhitespace`, `notationDecl`,
`processingInstruction`, `resolveEntity`, `skippedEntity`, `startPrefixMapping`,
`unparsedEntityDecl`, `warning`

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`,
`wait`, `wait`

Field Detail

`out`

`private static java.io.Writer out`

`indentString`

`private java.lang.String indentString`

`indentLevel`

private int **indentLevel**

parsedFile

private java.lang.String[] **parsedFile**

file

private java.io.File **file**

fileTags

private dbsection.FileTags **fileTags**

secElementsFound

private boolean **secElementsFound**

usSecElementFound

private boolean **usSecElementFound**

nonusSecElementFounf

private boolean **nonusSecElementFounf**

secCategory

private java.lang.String **secCategory**

index

private int **index**

Constructor Detail

Echo24

public **Echo24()**

The default constructor of the class

Method Detail

parseFile

public dbsection.FileTags **parseFile** (java.lang.String filename)

This method parses the file by using one of the SAXParsers from the SAXParserFactory

Parameters:

`filename` - The name of the file to be parsed

Returns:

A FileTags object containing all the security tags of the passed file

Throws:

`java.lang.Throwable` - In case of an error during the process

mainNew

`public dbsection.FileTags mainNew(java.lang.String argv)`

This method parses the file in the passed URL by using one of the SAXParsers from the SAXParserFactory

Returns:

A FileTags object containing all the security tags of the passed file

Throws:

`java.lang.Throwable` - In case of an error during the process

getFile

`public java.io.File getFile()`

Gets the file contained in the object

Returns:

The file contained in the object

getParsedFile

`public java.lang.String[] getParsedFile()`

Gets the parsed file contained in the object

Returns:

The file parsed contained in the object in an array of String

setDocumentLocator

`public void setDocumentLocator(org.xml.sax.Locator l)`

Specified by:

`setDocumentLocator` in interface `org.xml.sax.ContentHandler`

Overrides:

`setDocumentLocator` in class `org.xml.sax.helpers.DefaultHandler`

startDocument

`public void startDocument()`

throws `org.xml.sax.SAXException`

Specified by:

`startDocument` in interface `org.xml.sax.ContentHandler`

Overrides:

`startDocument` in class `org.xml.sax.helpers.DefaultHandler`
`org.xml.sax.SAXException`

endDocument

public void **endDocument**()
throws org.xml.sax.SAXException

Specified by:

endDocument in interface org.xml.sax.ContentHandler

Overrides:

endDocument in class org.xml.sax.helpers.DefaultHandler
org.xml.sax.SAXException

startElement

public void **startElement**(java.lang.String namespaceURI,
java.lang.String lName,
java.lang.String qName,
org.xml.sax.Attributes attrs)
throws org.xml.sax.SAXException

Specified by:

startElement in interface org.xml.sax.ContentHandler

Overrides:

startElement in class org.xml.sax.helpers.DefaultHandler
org.xml.sax.SAXException

endElement

public void **endElement**(java.lang.String namespaceURI,
java.lang.String sName,
java.lang.String qName)
throws org.xml.sax.SAXException

Specified by:

endElement in interface org.xml.sax.ContentHandler

Overrides:

endElement in class org.xml.sax.helpers.DefaultHandler
org.xml.sax.SAXException

characters

public void **characters**(char[] buf,
int offset,
int len)
throws org.xml.sax.SAXException

Specified by:

characters in interface org.xml.sax.ContentHandler

Overrides:

characters in class org.xml.sax.helpers.DefaultHandler
org.xml.sax.SAXException

emit

```
private void emit(java.lang.String s)
    throws org.xml.sax.SAXException
org.xml.sax.SAXException
```

nl

```
private void nl()
    throws org.xml.sax.SAXException
org.xml.sax.SAXException
```

dbsection

Class FileTags

java.lang.Object

|

+dbsection.FileTags

public class **FileTags**

extends java.lang.Object

The class FileTags is responsible to create an object containing all the security elements and attributes of the specific file that belongs

Field Summary

(package private) java.io.File	<u>file</u>
(package private) java.lang.String	<u>filename</u>
(package private) java.lang.String[]	<u>secAttribute</u>
(package private) java.lang.String[]	<u>secAttributeValue</u>

(package private) java.lang.String[]	<u>secElement</u>
(package private) java.lang.String[]	<u>secElementValue</u>

Constructor Summary

[FileTags](#)(java.io.File newFile)

The default constructor of the class receives the passed File object and initializes the parameter for itself

Method Summary

void	<u>checkAttribute</u> (java.lang.String category, java.lang.String newElement, java.lang.String newAttribute, java.lang.String newAttributeValue) Checks the passed attribute if it is a security attribute
boolean	<u>checkElement</u> (java.lang.String category, java.lang.String newElement, java.lang.String newElementValue) Checks the passed element if it is a security element
java.lang.String[]	<u>getSecAttributes</u> () Gets file's security attributes contained in the object
java.lang.String[]	<u>getSecAttributeValues</u> () Gets file's security attributes values contained in the object
private void	<u>setFileName</u> (java.lang.String newFileName) Sets the name of the file

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

filename

java.lang.String **filename**

secElement

java.lang.String[] **secElement**

secElementValue

java.lang.String[] **secElementValue**

secAttribute

java.lang.String[] **secAttribute**

secAttributeValue

java.lang.String[] **secAttributeValue**

file

java.io.File **file**

Constructor Detail

FileTags

public **FileTags**(java.io.File newFile)

The default constructor of the class receives the passed File object and initializes the parameter for itself

Parameters:

newFile - The File object on which that FileTags object is referred to

Method Detail

setFileName

private void **setFileName**(java.lang.String newFileName)

Sets the name of the file

Parameters:

newFileName - The new filename

Returns:

void

getSecAttributes

public java.lang.String[] **getSecAttributes**()

Gets file's security attributes contained in the object

Returns:

The file's security attributes contained in the object

getSecAttributeValue

public java.lang.String[] **getSecAttribute Values**()

Gets file's security attributes values contained in the object

Returns:

The file's security attributes values contained in the object

checkElement

```
public boolean checkElement(java.lang.String category,  
                             java.lang.String newElement,  
                             java.lang.String newElementValue)
```

Checks the passed element if it is a security element

Parameters:

category - The element's category

newElement - The element to be checked

newElementValue - The element's value

Returns:

A boolean true if the element is a security element

checkAttribute

```
public void checkAttribute(java.lang.String category,  
                           java.lang.String newElement,  
                           java.lang.String newAttribute,  
                           java.lang.String newAttributeValue)
```

Checks the passed attribute if it is a security attribute

Parameters:

category - The attribute's category

newElement - The element containing the attribute

newAttribute - The attribute to be checked

newAttributeValue - The attribute's value

Returns:

void

dbsection
Class FindUrl

```
java.lang.Object
|
+-javax.servlet.GenericServlet
|
+-javax.servlet.http.HttpServlet
|
+-dbsection.FindUrl
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

public class **FindUrl**

extends javax.servlet.http.HttpServlet

implements java.io.Serializable

The FindUrl class defines a servlet that is responsible to connect the system with the requested server, to retrieve the releasable files from that server and to decide which of those files the user is allowed to access

See Also:

[Serialized Form](#)

Field Summary	
private java.lang.String	authentication
private boolean	decision

private java.io.File	<u>file</u>
private static dbsection.FileTags	<u>fileTags</u>
private java.lang.String	<u>localServer</u>
private static dbsection.SecMetadataManager	<u>manager</u>
private java.util.Vector	<u>matchedFiles</u>
private java.lang.String	<u>npsServer</u>
private java.io.PrintWriter	<u>out</u>
private java.lang.String	<u>permission</u>
private javax.servlet.http.HttpServletRequest	<u>req</u>
private java.lang.String	<u>ReqFilename</u>
private javax.servlet.http.HttpServletResponse	<u>res</u>
private java.lang.String	<u>selectedUrl</u>
private java.lang.String	<u>selUrl</u>
private java.lang.String	<u>server</u>
private java.util.Vector	<u>serverFoundFiles</u>
private java.lang.String	<u>thisServer</u>
private java.lang.String	<u>thisServerPassword</u>
private dbsection.UserBean	<u>user</u>

Fields inherited from class javax.servlet.http.HttpServlet

--

Fields inherited from class javax.servlet.GenericServlet

--

Constructor Summary

[FindUrl\(\)](#)

Method Summary

private java.util.Vector	checkIfExist (java.lang.String fileName)
private void	checkingFile (java.lang.String filename)
void	doGet (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Processing user's GET request by simply passing the control to the doPost.
void	doPost (javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res) Processing user's POST request.
private void	exitLinks ()
private java.lang.String[]	findVersion (java.lang.String fullName)
private void	getConnected (java.lang.String server)
private void	printData ()
private void	startSession () Starts a new session after a customer is correctly authenticated.

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

user

private dbsection.UserBe an **user**

out

private java.io.PrintWriter **out**

req

private javax.servlet.http.HttpServletRequest **req**

res

private javax.servlet.http.HttpServletResponse **res**

selUrl

private java.lang.String **selUrl**

selectedUrl

private java.lang.String **selectedUrl**

permission

private java.lang.String **permission**

authentication

private java.lang.String **authentication**

fileTags

private static dbsection.FileTags **fileTags**

manager

private static dbsection.SecMetadataManager **manager**

decision

private boolean **decision**

file

private java.io.File **file**

thisServer

private java.lang.String **thisServer**

thisServerPassword

private java.lang.String **thisServerPassword**

serverFoundFiles

private java.util.Vector **serverFoundFiles**

matchedFiles

private java.util.Vector **matchedFiles**

npsServer

private java.lang.String **npsServer**

localServer

private java.lang.String **localServer**

server

private java.lang.String **server**

ReqFilename

private java.lang.String **ReqFilename**

Constructor Detail

FindUrl

public **FindUrl()**

Method Detail

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,  
                 javax.servlet.http.HttpServletResponse response)  
    throws javax.servlet.ServletException,  
           java.io.IOException
```

Processing user's GET request by simply passing the control to the doPost.

Overrides:

doGet in class javax.servlet.http.HttpServlet

Parameters:

request - The client's request.

response - The response to the client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,  
                  javax.servlet.http.HttpServletResponse res)  
    throws java.io.IOException
```

Processing user's POST request.

Overrides:

doPost in class javax.servlet.http.HttpServlet

Parameters:

req - the client request.

res - the response to client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

getConnected

```
private void getConnected(java.lang.String server)
```

startSession

```
private void startSession()  
    throws java.lang.Exception
```

Starts a new session after a customer is correctly authenticated.

Returns:

void.

Throws:

Exception.
java.lang.Exception

checkIfExists

private java.util.Vector **checkIfExists**(java.lang.String fileName)

findVersion

private java.lang.String[] **findVersion**(java.lang.String fullName)

checkingFile

private void **checkingFile**(java.lang.String filename)

exitLinks

private void **exitLinks**()

printData

private void **printData**()

dbsection

Class ImportFile

```
java.lang.Object
|
+-javax.servlet.GenericServlet
|
+-javax.servlet.http.HttpServlet
|
+-dbsection.ImportFile
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

public class **ImportFile**

extends javax.servlet.http.HttpServlet

The ImportFile class defines a servlet that is responsible to create a web page to help user input the file and the server where that file is located in order to open it.

See Also:

[Serialized Form](#)

Field Summary

private java.util.Vector	avaFiles
private dbsection.UserBean	freshuser
private java.io.PrintWriter	out
private javax.servlet.http.HttpServletRequest	req

private javax.servlet.http.HttpServletResponse	res

Fields inherited from class javax.servlet.http.HttpServlet

Fields inherited from class javax.servlet.GenericServlet

Constructor Summary

[ImportFile](#) ()

Method Summary

void	doPost (javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res) Processing user's POST request.
private void	exitLinks () Displays the available links to direct control in another servlet, in case the user wants to cancel this procedure
private void	inputUrl () Creates the necessary radio buttons for the server's selection, the text field to get the file and the buttons to clear the field or to submit the filename
private void	printData () Prints the user's personal and Security data retrieved from the database that contained in the UserBean object

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doGet, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,

```
wait, wait, wait
```

Field Detail

freshuser

private dbsection.UserBean **freshuser**

avaFiles

private java.util.Vector **avaFiles**

out

private java.io.PrintWriter **out**

req

private javax.servlet.http.HttpServletRequest **req**

res

private javax.servlet.http.HttpServletResponse **res**

Constructor Detail

ImportFile

public **ImportFile** ()

Method Detail

doPost

public void **doPost**(javax.servlet.http.HttpServletRequest req,
 javax.servlet.http.HttpServletResponse res)
 throws java.io.IOException

Processing user's POST request.

Overrides:

doPost in class javax.servlet.http.HttpServlet

Parameters:

req - the client request.

res - the response to client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

exitLinks

private void **exitLinks()**
Displays the available links to direct control in another servlet, in case the user wants to cancel this procedure

Returns:
void.

inputUrl

private void **inputUrl()**
Creates the necessary radio buttons for the server's selection, the text field to get the file and the buttons to clear the field or to submit the filename

Returns:
void.

printData

private void **printData()**
Prints the user's personal and Security data retrieved from the database that contained in the UserBean object

Returns:
void.

dbsection
Class ImportToSave

```
java.lang.Object
|
+-javax.servlet.GenericServlet
|
+-javax.servlet.http.HttpServlet
|
+-dbsection.ImportToSave
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

```
public class ImportToSave
extends javax.servlet.http.HttpServlet
implements java.io.Serializable
```

The ImportToSave class defines a servlet that is responsible to create a web page to help user input the file to be saved, as well as some information related to the file

See Also:

[Serialized Form](#)

Field Summary	
private java.util.Vector	avaFiles
private java.lang.String[]	classifi
private java.lang.String[]	countries

private java.io.PrintWriter	<u>out</u>
private javax.servlet.http.HttpServletRequest	<u>req</u>
private javax.servlet.http.HttpServletResponse	<u>res</u>
private dbsection.User bean	<u>sessionuser</u>

Fields inherited from class javax.servlet.http.HttpServlet

Fields inherited from class javax.servlet.GenericServlet

Constructor Summary

[ImportToSave](#) ()

Method Summary

void	<u>doGet</u> (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Processing user's GET request by simply passing the control to the doPost.
void	<u>doPost</u> (javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res) Processing user's POST request.
private java.lang.String	<u>dropDownClass</u> (java.lang.String useClass) This method creates the allowed options in the drop down list for the classification selection of the file
private java.lang.String	<u>dropDownCountry</u> (java.lang.String userCountry) This method creates the allowed options in the drop down list for the releasable countries of the file
private void	<u>exitLinks</u> () Displays the available links to direct control in another servlet, in case the user wants to cancel this procedure
private void	<u>inputFile</u> () Creates the necessary text field to help the user to input the file and the buttons to clear the field or to submit the filename

void	private printStandardMetadata () Prints the user's personal and security data that will appended to the file as its metadata.
void	private tableRow (java.lang.String firstColumn, java.lang.String secondColumn, java.lang.String thirdColumn, java.lang.String forthColumn) This method displays the passed parameters in a table row with specific dimensions

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

sessionuser

private dbsection.UserBean **sessionuser**

avaFiles

private java.util.Vector **avaFiles**

out

private java.io.PrintWriter **out**

req

private javax.servlet.http.HttpServletRequest **req**

res

private javax.servlet.http.HttpServletResponse **res**

classifi

private java.lang.String[] **classifi**

countries

private java.lang.String[] countries

Constructor Detail

ImportToSave

public **ImportToSave**()

Method Detail

doGet

public void **doGet**(javax.servlet.http.HttpServletRequest request,
 javax.servlet.http.HttpServletResponse response)
 throws javax.servlet.ServletException,
 java.io.IOException

Processing user's GET request by simply passing the control to the doPost.

Overrides:

doGet in class javax.servlet.http.HttpServlet

Parameters:

request - The client's request.

response - The response to the client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

doPost

public void **doPost**(javax.servlet.http.HttpServletRequest req,
 javax.servlet.http.HttpServletResponse res)
 throws javax.servlet.ServletException,
 java.io.IOException

Processing user's POST request.

Overrides:

doPost in class javax.servlet.http.HttpServlet

Parameters:

req - the client request.

res - the response to client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

exitLinks

private void **exitLinks()**

Displays the available links to direct control in another servlet, in case the user wants to cancel this procedure

Returns:

void.

inputFile

private void **inputFile()**

Creates the necessary text field to help the user to input the file and the buttons to clear the field or to submit the filename

Returns:

void.

printStandardMetadata

private void **printStandardMetadata()**

Prints the user's personal and security data that will be appended to the file as its metadata.

Returns:

void.

dropDownClass

private java.lang.String **dropDownClass**(java.lang.String userClass)

This method creates the allowed options in the drop down list for the classification selection of the file

Parameters:

`userClass` - The classification of the user, which will be the maximum allowed for the file

Returns:

The allowed classifications for the file

dropDownCountry

private java.lang.String **dropDownCountry**(java.lang.String userCountry)

This method creates the allowed options in the drop down list for the releasable countries of the file

Returns:

The allowed countries for the file

tableRow

private void **tableRow**(java.lang.String firstColumn,

```

        java.lang.String secondColumn,
        java.lang.String thirdColumn,
        java.lang.String forthColumn)

```

This method displays the passed parameters in a table row with specific dimensions

Parameters:

```

firstColumn - The first column of the row
secondColumn - The second column of the row
thirdColumn - The third column of the row

```

Returns:

```

void

```

dbsection

Class Login

```

java.lang.Object
|
+-javax.servlet.GenericServlet
|
+-javax.servlet.http.HttpServlet
|
+-dbsection.Login

```

All Implemented Interfaces:

```

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

```

public class Login

```

extends javax.servlet.http.HttpServlet

```

The class Login defines a servlet that is responsible to authenticate users trying to login. Upon positive authentication, a new user session is created with two attached beans: the customer bean and the database connection module.

See Also:

[Serialized Form](#)

Field Summary	
private java.util.Vector	avaFiles
private java.sql.Connection	connection
private int	CONNECTION_ERROR
private java.lang.String	curServiceClass
private java.lang.String	curServiceName

private java.lang.String	<u>curServicePos</u>
private java.lang.String	<u>curUserCountryId</u>
private int	<u>curUserId</u>
private dbsection.DBConnectionBean	<u>dbBean</u>
private java.lang.Exception	<u>dbe</u>
private int	<u>DRIVER_ERROR</u>
private int	<u>INVALID_USER</u>
private java.io.PrintWriter	<u>out</u>
private int	<u>QUERY_ERROR</u>
private int	<u>QUERY_ERROR_EXC</u>
private javax.servlet.http.HttpServletRequest	<u>request</u>
private javax.servlet.http.HttpServletResponse	<u>response</u>
private java.sql.ResultSet	<u>rs</u>
private java.sql.ResultSet	<u>rs2</u>
private dbsection.UserBean	<u>user</u>

Fields inherited from class javax.servlet.http.HttpServlet

--

Fields inherited from class javax.servlet.GenericServlet

--

Constructor Summary	
Login()	

Method Summary	
private dbsection.UserBean	createUser() Creates a new userBean object from the information retrieved from the database.
void	doGet (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Processing user's GET request.
void	doPost (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Processing user's POST request by simply passing the control to the doGet.
private void	exitPoint (int exitCondition) Defines several types of exit conditions depending on the specified error
private void	findFiles (java.lang.String service, java.lang.String position, java.lang.String clearance, java.lang.String country) Finds the files that are available to a specific user according to his/her security authorizations
private void	findNamePos (java.lang.String name, java.lang.String password) Finds the Security authorizations of a valid user
void	init (javax.servlet.ServletConfig config) One-time initialization of the servlet.
private void	startSession () Starts a new session after a user is successfully authenticated.
private void	validateUser (java.lang.String name, java.lang.String password) Authenticates a user with the given username & password

Methods inherited from class javax.servlet.http.HttpServlet
doDelete, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet
destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

DRIVER_ERROR

private final int **DRIVER_ERROR**

See Also:

[Constant Field Values](#)

CONNECTION_ERROR

private final int **CONNECTION_ERROR**

See Also:

[Constant Field Values](#)

QUERY_ERROR

private final int **QUERY_ERROR**

See Also:

[Constant Field Values](#)

QUERY_ERROR_EXC

private final int **QUERY_ERROR_EXC**

See Also:

[Constant Field Values](#)

INVALID_USER

private final int **INVALID_USER**

See Also:

[Constant Field Values](#)

request

private javax.servlet.http.HttpServletRequest **request**

response

private javax.servlet.http.HttpServletResponse **response**

connection

private java.sql.Connection **connection**

out

private java.io.PrintWriter **out**

rs

private java.sql.ResultSet **rs**

rs2

private java.sql.ResultSet **rs2**

dbBean

private dbsection.DBConnectionBean **dbBean**

curServiceName

private java.lang.String **curServiceName**

curServicePos

private java.lang.String **curServicePos**

curServiceClass

private java.lang.String **curServiceClass**

curUserCountryId

private java.lang.String **curUserCountryId**

curUserId

private int **curUserId**

dbe

private java.lang.Exception **dbe**

user

private dbsection.UserBean **user**

avaFiles

private java.util.Vector **avaFiles**

Constructor Detail

Login

public **Login**()

Method Detail

init

public void **init**(javax.servlet.ServletConfig config)

throws javax.servlet.ServletException

One-time initialization of the servlet. If the Connection Module is not yet initiated, it ensures its creation.

Specified by:

init in interface javax.servlet.Servlet

Overrides:

init in class javax.servlet.GenericServlet

Parameters:

config - local server configuration parameters.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

doPost

public void **doPost**(javax.servlet.http.HttpServletRequest request,

javax.servlet.http.HttpServletResponse response)

throws javax.servlet.ServletException,

java.io.IOException

Processing user's POST request by simply passing the control to the doGet.

Overrides:

doPost in class javax.servlet.http.HttpServlet

Parameters:

request - The client's request.

response - The response to the client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,
                 javax.servlet.http.HttpServletResponse response)
    throws javax.servlet.ServletException,
           java.io.IOException
Processing user's GET request.
Overrides:
doGet in class javax.servlet.http.HttpServlet
Parameters:
request - the client request.
response - the response to client.
Returns:
void.
Throws:
javax.servlet.ServletException - In case of a servlet error encountered
java.io.IOException - In case of a I/O error encountered
```

validateUser

```
private void validateUser(java.lang.String name,
                          java.lang.String password)
Authenticates a user with the given username & password
Parameters:
name - The user's username.
password - The user's password.
Returns:
void.
Throws:
javax.servlet.ServletException - In case of a servlet error encountered
java.io.IOException - In case of a servlet error encountered
```

findNamePos

```
private void findNamePos(java.lang.String name,
                          java.lang.String password)
Finds the Security authorizations of a valid user
Parameters:
name - The user's username.
password - The user's password.
Returns:
void.
Throws:
javax.servlet.ServletException - In case of a servlet error encountered
java.io.IOException - In case of a servlet error encountered
```

findFiles

```
private void findFiles(java.lang.String service,  
                      java.lang.String position,  
                      java.lang.String clearance,  
                      java.lang.String country)
```

Finds the files that are available to a specific user according to his/her security authorizations

Parameters:

`service` - The user's service

`position` - The user's position in his or her service

`clearance` - The user's clearance (Unclassified, Classified etc.)

`country` - The user's country

Returns:

void.

Throws:

`javax.servlet.ServletException` - In case of a servlet error encountered

`java.io.IOException` - In case of an I/O error encountered

`startSession`

```
private void startSession()  
    throws java.lang.Exception
```

Starts a new session after a user is successfully authenticated.

Returns:

void

Throws:

`java.lang.Exception` - In case of an invalid situation is encountered

`createUser`

```
private dbsec tion.UserBean createUser()  
    throws java.lang.Exception
```

Creates a new userBean object from the information retrieved from the database.

Returns:

A UserBean object containing all the user's data is returned.

Throws:

`java.lang.Exception` - In the case of an invalid data

`exitPoint`

```
private void exitPoint(int exitCondition)
```

Defines several types of exit conditions depending on the specified error

Parameters:

`exitCondition` - That integer specifies the error causes the exit.

Returns:

void.

dbsection
Class Logout

```

java.lang.Object
|
+-javax.servlet.GenericServlet
|
+-javax.servlet.http.HttpServlet
|
+-dbsection.Logout

```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

public class **Logout**

extends javax.servlet.http.HttpServlet

This class defines a servlet that logout a user and end his/her session.

See Also:

[Serialized Form](#)

Field Summary	
private dbsection.CustomerBean	<u>customer</u>
private dbsection.DBConnectionBean	<u>dbBean</u>
private int	<u>ERROR_SESSION</u>
private java.io.PrintWriter	<u>out</u>
private java.lang.String	<u>output</u>
private javax.servlet.http.HttpServletRequest	<u>request</u>

private javax.servlet.http.HttpServletRespons e	response
private java.sql.ResultSet	rs

Fields inherited from class javax.servlet.http.HttpServlet

Fields inherited from class javax.servlet.GenericServlet

Constructor Summary

[Logout\(\)](#)

Method Summary

void	doGet (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Processing user's GET request.
void	doPost (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Processing user's POST request by simply passing the control to the doGet.
private void	exitLinks () Displays the available links to direct control in another page, in case the user wants to discontinue the session
private void	processRequest () Processing the request for logging the user out and terminating the current user session.

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig,
getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,

wait, wait, wait

Field Detail

ERROR_SESSION

private final int **ERROR_SESSION**

See Also:

[Constant Field Values](#)

request

private javax.servlet.http.HttpServletRequest **request**

response

private javax.servlet.http.HttpServletResponse **response**

out

private java.io.PrintWriter **out**

rs

private java.sql.ResultSet **rs**

dbBean

private dbsection.DBConnectionBean **dbBean**

customer

private dbsection.CustomerBean **customer**

output

private java.lang.String **output**

Constructor Detail

Logout

public **Logout()**

Method Detail

doPost

public void **doPost**(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)

throws javax.servlet.ServletException,
java.io.IOException

Processing user's POST request by simply passing the control to the doGet.

Overrides:

doPost in class javax.servlet.http.HttpServlet

Parameters:

request - The client's request.

response - The response to the client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

doGet

public void **doGet**(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException,
java.io.IOException

Processing user's GET request.

Overrides:

doGet in class javax.servlet.http.HttpServlet

Parameters:

request - the client request.

response - the response to client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

processRequest

private void **processRequest**()

Processing the request for logging the user out and terminating the current user session.

Returns:

void

exitLinks

private void **exitLinks**()

Displays the available links to direct control in another page, in case the user wants to discontinue the session

Returns:

void.

dbsection

Class MetaTags

java.lang.Object

|

+dbsection.MetaTags

public class **MetaTags**

extends java.lang.Object

The MetaTags class codes the tags of a file as an object

Field Summary

(package private) int	index
(package private) java.lang.String[]	parsedFile

Constructor Summary

MetaTags ()
The default constructor of the class initializes the variables.

Method Summary

void	add (java.lang.String newString) Adds a new tag passed as the parameter
java.lang.String[]	getParsedFile () Gets the tags of the parsed file

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
--

Field Detail

parsedFile

java.lang.String[] **parsedFile**

index

int **index**

Constructor Detail

MetaTags

public **MetaTags**()

The default constructor of the class initializes the variables.

Method Detail

getParsedFile

public java.lang.String[] **getParsedFile**()

Gets the tags of the parsed file

Returns:

An array of String representing the tags of the file

add

public void **add**(java.lang.String newString)

Adds a new tag passed as the parameter

Parameters:

newString - The new tag to be added

Returns:

void

dbsection
Class OpenFile

```
java.lang.Object
|
+-javax.servlet.GenericServlet
|
+-javax.servlet.http.HttpServlet
|
+-dbsection.OpenFile
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

public class **OpenFile**

extends javax.servlet.http.HttpServlet

The class OpenFile defines a servlet that is responsible to open any file a user may select either the file is saved locally or in another server

See Also:

[Serialized Form](#)

Field Summary	
private java.lang.String	action
private java.util.Vector	avaCLFiles
private java.util.Vector	avaFiles
private java.util.Vector	avaSEFiles
private java.util.Vector	avaTSFiles
private java.util.Vector	avaUNFiles

private java.lang.String	<u>classif</u>
private java.lang.String[]	<u>classification</u>
private java.lang.String[]	<u>date</u>
private dbsection.DBConnectionBean	<u>dbBean</u>
private dbsection.UserBean	<u>newuser</u>
private java.lang.String	<u>otherServer</u>
private java.io.PrintWriter	<u>out</u>
private javax.servlet.http.HttpServletRequest	<u>req</u>
private javax.servlet.http.HttpServletResponse	<u>res</u>
private java.lang.String	<u>selfFile</u>

Fields inherited from class javax.servlet.http.HttpServlet

Fields inherited from class javax.servlet.GenericServlet

Constructor Summary

[OpenFile](#) ()

Method Summary

vc id	<u>doGet</u> (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Processing user's GET request by simply passing the control to the doPost.
vc id	<u>doPost</u> (javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)

	Processing user's POST request.
private java.util.Vector	<u>findFiles</u> (java.lang.String service, java.lang.String position, java.lang.String clearance, java.lang.String country) Finds the files that are available to a specific user according to his security authorizations
private java.lang.String[]	<u>findVersion</u> (java.lang.String fullName) This method analyses the filename to extract the date and the time, the file saved in the system which actually represents the version of the file.
private int	<u>getClassLevel</u> (java.lang.String classification) Converts user's classification to a number
private void	<u>openConnection</u> () Opens a connection with the database to retrieve the files that are available to the user
private void	<u>printAvaFiles</u> () This method gets the available files through the findFiles method and display them to the user according to his/her classification
private void	<u>printData</u> () Prints the user's personal and Security data retrieved from the database that contained in the UserBean object
private void	<u>printServerAvaFiles</u> () This method gets the available files passed from the associated server and present them to the user according to his/her classification
private void	<u>startEditor</u> (java.lang.String classLevel, java.lang.String sFile) Calls the applet that will in turn load and initiate the applet-based XML editor

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

newuser

private dbsection.UserBean **newuser**

avaUNFiles

private java.util.Vector **avaUNFiles**

avaCLFiles

private java.util.Vector **avaCLFiles**

avaSEFiles

private java.util.Vector **avaSEFiles**

avaTSFiles

private java.util.Vector **avaTSFiles**

out

private java.io.PrintWriter **out**

dbBean

private dbsection.DBConnectionBean **dbBean**

classification

private java.lang.String[] **classification**

date

private java.lang.String[] **date**

req

private javax.servlet.http.HttpServletRequest **req**

res

private javax.servlet.http.HttpServletResponse **res**

selFile

private java.lang.String **selfFile**

classif

private java.lang.String **classif**

action

private java.lang.String **action**

otherServer

private java.lang.String **otherServer**

avaFiles

private java.util.Vector **avaFiles**

Constructor Detail

OpenFile

public **OpenFile** ()

Method Detail

doGet

public void **doGet**(javax.servlet.http.HttpServletRequest request,
 javax.servlet.http.HttpServletResponse response)
 throws javax.servlet.ServletException,
 java.io.IOException

Processing user's GET request by simply passing the control to the doPost.

Overrides:

doGet in class javax.servlet.http.HttpServlet

Parameters:

request - The client's request.

response - The response to the client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

doPost

public void **doPost**(javax.servlet.http.HttpServletRequest req,
 javax.servlet.http.HttpServletResponse res)

throws java.io.IOException

Processing user's POST request.

Overrides:

doPost in class javax.servlet.http.HttpServlet

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

startEditor

private void **startEditor**(java.lang.String classLevel,
java.lang.String sFile)

Calls the applet that will in turn load and initiate the applet-based XML editor

Parameters:

classLevel - The classification level of the user that will help find the directory
where the file is saved.

sFile - The name of the file that the editor will open

Returns:

void.

getClassLevel

private int **getClassLevel**(java.lang.String classification)

Converts user's classification to a number

Parameters:

classification - The classification level of the user

Returns:

An integer number representing the user's classification

printServerAvaFiles

private void **printServerAvaFiles**()

This method gets the available files passed from the associated server and present
them to the user according to his/her classification

Returns:

void

printAvaFiles

private void **printAvaFiles**()

This method gets the available files through the findFiles method and display
them to the user according to his/her classification

Returns:

void

findVersion

private java.lang.String[] **findVersion**(java.lang.String fullName)

This method analyses the filename to extract the date and the time, the file saved in the system which actually represents the version of the file.

Parameters:

fullName - The complete name of the file

Returns:

An array of string containing the date and the time

findFiles

private java.util.Vector **findFiles**(java.lang.String service,
java.lang.String position,
java.lang.String clearance,
java.lang.String country)

Finds the files that are available to a specific user according to his security authorizations

Parameters:

service - The user's service

position - The user's position in his or her service

clearance - The user's clearance (Unclassified, Classified etc.)

country - The user's country

Returns:

void.

Throws:

java.lang.Exception - In case of a error encountered

openConnection

private void **openConnection**()

Opens a connection with the database to retrieve the files that are available to the user

Returns:

void

printData

private void **printData**()

Prints the user's personal and Security data retrieved from the database that contained in the UserBean object

Returns:

void.

dbsection
Class SaveFile

```
java.lang.Object
|
+-javax.servlet.GenericServlet
|
+-javax.servlet.http.HttpServlet
|
+-dbsection.SaveFile
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

public class **SaveFile**
extends javax.servlet.http.HttpServlet
implements java.io.Serializable

The SaveFile class is responsible to upload the file from the user's machine to a temporary area of the system and to check and parse the file

See Also:

[Serialized Form](#)

Field Summary	
private boolean	<u>decision</u>
private java.lang.String	<u>desiredClass</u>
private java.lang.String	<u>desiredCountry</u>
private java.lang.String	<u>ending</u>
private java.lang.String	<u>errorState</u>
private java.io.File	<u>file</u>
private java.lang.String	<u>filename</u>

private java.lang.String[]	<u>fileSecAttr</u>
(package private) static dbsection.FileTags	<u>fileTags</u>
private java.lang.String	<u>fiuo</u>
private java.lang.String	<u>fouo</u>
(package private) static dbsection.SecMetadataManager	<u>manager</u>
private int	<u>MAXIMUM_FILE_LINES</u>
private java.io.PrintWriter	<u>out</u>
private javax.servlet.http.HttpServletRequest	<u>req</u>
private javax.servlet.http.HttpServletResponse	<u>res</u>
private java.io.File	<u>selectedFile</u>
private java.lang.String	<u>tempFile</u>
private java.lang.String	<u>uploaded</u>
private dbsection.UserBean	<u>user</u>

Fields inherited from class javax.servlet.http.HttpServlet

Fields inherited from class javax.servlet.GenericServlet

Constructor Summary

[SaveFile](#) ()

Method Summary	
private void	askConfirm (java.lang.String status) In the cases where the status found is "upgrade" or "downgrade" this method displays the appropriate message
private void	checkingFile (java.lang.String selFile) This method passes the file to the Echo24 class which parses the file and returns a filetags object This object and the userBean object are passed to the security manager for the decision.
void	doPost (javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res) Processing user's POST request.
private void	errorGoBack (java.lang.String error) Defines several types of error conditions depending on the specified error
private void	exitLinks () Displays the available links to direct control in another servlet, in case the user wants to cancel this procedure
private java.lang.String	findFilename (java.lang.String stringFile) This method extracts the name of the file from incoming stream, and checks if it is a valid XML filename
private void	printData () Prints the user's personal and Security data retrieved from the database that contained in the UserBean object
private java.lang.String	uploadFile () Uploads the file to a temporary position for further processing

Methods inherited from class javax.servlet.http.HttpServlet
doDelete, doGet, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet
destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

user

private dbsection.UserBean **user**

req

private javax.servlet.http.HttpServletRequest **req**

res

private javax.servlet.http.HttpServletResponse **res**

uploaded

private java.lang.String **uploaded**

selectedFile

private java.io.File **selectedFile**

file

private java.io.File **file**

out

private java.io.PrintWriter **out**

MAXIMUM_FILE_LINES

private int **MAXIMUM_FILE_LINES**

decision

private boolean **decision**

desiredClass

private java.lang.String **desiredClass**

desiredCountry

private java.lang.String **desiredCountry**

fouo

private java.lang.String **fouo**

fiuo

private java.lang.String **fiuo**

tempFile

private java.lang.String **tempFile**

filename

private java.lang.String **filename**

ending

private java.lang.String **ending**

errorState

private java.lang.String **errorState**

fileSecAttr

private java.lang.String[] **fileSecAttr**

fileTags

static dbsection.FileTags **fileTags**

manager

static dbsection.SecMetadataManager **manager**

Constructor Detail

SaveFile

public **SaveFile()**

Method Detail

doPost

public void **doPost**(javax.servlet.http.HttpServletRequest req,
 javax.servlet.http.HttpServletResponse res)

throws java.io.IOException

Processing user's POST request.

Overrides:

doPost in class javax.servlet.http.HttpServlet

Returns:

void.

Throws:

`javax.servlet.ServletException` - In case of a servlet error encountered

`java.io.IOException` - In case of a I/O error encountered

uploadFile

private java.lang.String **uploadFile**()
throws `java.io.IOException`

Uploads the file to a temporary position for further processing

Returns:

If everything was done successfully returns the name of the temporary file

Throws:

`java.io.IOException` - In case a communication exception is thrown

findFilename

private java.lang.String **findFilename**(java.lang.String stringFile)

This method extracts the name of the file from incoming stream, and checks if it is a valid XML filename

Parameters:

`stringFile` - The complete name of the file

Returns:

The name of the file found

errorGoBack

private void **errorGoBack**(java.lang.String error)

Defines several types of error conditions depending on the specified error

Parameters:

`error` - The error specifies the cause of the problem.

Returns:

void.

checkingFile

private void **checkingFile**(java.lang.String selfFile)

This method passes the file to the Echo24 class which parses the file and returns a filetags object This object and the userBean object are passed to the security manager for the decision.

Parameters:

`selfFile` - the selected file to be parsed

Returns:

void.

askConfirm

private void **askConfirm**(java.lang.String status)

In the cases where the status found is "upgrade" or "downgrade" this method displays the appropriate message

Parameters:

status - The results of the tag's comparison

Returns:

void.

exitLinks

private void **exitLinks**()

Displays the available links to direct control in another servlet, in case the user wants to cancel this procedure

Returns:

void.

printData

private void **printData**()

Prints the user's personal and Security data retrieved from the database that contained in the UserBean object

Returns:

void.

dbsection
Class SecMetadataManager

```
java.lang.Object
|
+-dbsection.SecMetadataManager
```

```
public class SecMetadataManager
extends java.lang.Object
```

The class SecMetadataManager is responsible to compare the received UserBean and FileTags objects and to produce the necessary decision according to the policy

Field Summary

private boolean	<u>approved</u>
private dbsection.FileTags	<u>fileTags</u>
private dbsection.UserBean	<u>userBean</u>

Constructor Summary

[SecMetadataManager](#)(dbsection.FileTags tags, dbsection.UserBean user)

The default constructor of the class receives the passed objects and initializes the variable of the decision as false

Method Summary

java.lang.String	<u>compareTags</u> (java.lang.String desiredClass, java.lang.String desiredCountry, java.lang.String fouo, java.lang.String fiuo) This method compares the tags of the file with those of the user, and returns the result.
private int	<u>getClassLevel</u> (java.lang.String classification) This method converts a classification level to a number to be easiest to compare
boolean	<u>makeDecision</u> ()

	This method makes the decision
--	--------------------------------

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

fileTags

private dbsection.FileTags **fileTags**

userBean

private dbsection.UserBean **userBean**

approved

private boolean **approved**

Constructor Detail

SecMetadataManager

public **SecMetadataManager**(dbsection.FileTags tags,
dbsection.UserBean user)

The default constructor of the class receives the passed objects and initializes the variable of the decision as false

Parameters:

tags - The FileTags object representing the tags of a specific file

Method Detail

makeDecision

public boolean **makeDecision**()

This method makes the decision

Returns:

A boolean variable containing the decision

compareTags

public java.lang.String **compareTags**(java.lang.String desiredClass,
java.lang.String desiredCountry,
java.lang.String fouo,
java.lang.String fiuo)

This method compares the tags of the file with those of the user, and returns the result.

Returns:

A String that can be upgrade, downgrade or normal

getClassLevel

private int **getClassLevel**(java.lang.String classification)

This method converts a classification level to a number to be easiest to compare

Returns:

An integer representing the classification level

dbsection

Class ServerLogin

java.lang.Object

|

+javax.servlet.GenericServlet

|

+javax.servlet.http.HttpServlet

|

+**dbsection.ServerLogin**

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

public class **ServerLogin**

extends javax.servlet.http.HttpServlet

The class ServerLogin defines a servlet that is responsible to authenticate associated servers trying to login. Upon positive authentication, all the names of the files that are available to that specific server are sent for further processing.

See Also:

[Serialized Form](#)

Field Summary	
java.util.Vector	avaFiles
private java.sql.Connection	connection
private int	CONNECTION_ERROR
private java.lang.String	curServiceClass
private java.lang.String	curServiceName
private java.lang.String	curServicePos
private java.lang.String	curUserCountryId

private int	<u>curUserId</u>
private dbsection.DBConnectionBean	<u>dbBean</u>
java.lang.Exception	<u>dbe</u>
private int	<u>DRIVER_ERROR</u>
private java.util.Enumeration	<u>enum</u>
private java.lang.String	<u>filename</u>
private int	<u>INVALID_USER</u>
private java.io.PrintWriter	<u>out</u>
private int	<u>QUERY_ERROR</u>
private int	<u>QUERY_ERROR_EXC</u>
private javax.servlet.http.HttpServletRequest	<u>request</u>
private javax.servlet.http.HttpServletResponse	<u>response</u>
private java.sql.ResultSet	<u>rs</u>
private java.sql.ResultSet	<u>rs2</u>
private java.lang.String[]	<u>savedFilename</u>
private java.lang.String	<u>server</u>
private java.lang.String	<u>serverName</u>
private javax.servlet.ServletOutputStream	<u>serverOut</u>
private java.lang.String	<u>serverPass</u>

dbsection.UserBean	user
--------------------	----------------------

Fields inherited from class javax.servlet.http.HttpServlet

Fields inherited from class javax.servlet.GenericServlet

Constructor Summary
ServerLogin()

Method Summary	
vc id	doGet (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Processing user's GET request.
vc id	doPost (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Processing user's POST request by simply passing the control to the doGet.
private java.util.Vector	findFiles (java.lang.String serverCode) Finds all the files that are releasable to that specific server according to the security authorizations contained in the database
private java.lang.String[]	findVersion (java.lang.String fullName) This method analyses the filename to extract the date and the time, the file saved in the system which actually represents the version of the file.
private void	openConnection () Opens a connection with the database to retrieve the available files of this user
private void	validateServer (java.lang.String name, java.lang.String pass) Authenticates a server with the given username & password

Methods inherited from class javax.servlet.http.HttpServlet
doDelete, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet
--

```
destroy, getInitParameter, getInitParameterNames, getServletConfig,  
getServletContext, getServletInfo, getServletName, init, init, log, log
```

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,  
wait, wait, wait
```

Field Detail

DRIVER_ERROR

private final int **DRIVER_ERROR**

See Also:

[Constant Field Values](#)

CONNECTION_ERROR

private final int **CONNECTION_ERROR**

See Also:

[Constant Field Values](#)

QUERY_ERROR

private final int **QUERY_ERROR**

See Also:

[Constant Field Values](#)

QUERY_ERROR_EXC

private final int **QUERY_ERROR_EXC**

See Also:

[Constant Field Values](#)

INVALID_USER

private final int **INVALID_USER**

See Also:

[Constant Field Values](#)

request

private javax.servlet.http.HttpServletRequest **request**

response

private javax.servlet.http.HttpServletResponse **response**

connection

private java.sql.Connection **connection**

out

private java.io.PrintWriter **out**

serverOut

private javax.servlet.ServletOutputStream **serverOut**

rs

private java.sql.ResultSet **rs**

rs2

private java.sql.ResultSet **rs2**

dbBean

private dbsection.DBConnectionBean **dbBean**

curServiceName

private java.lang.String **curServiceName**

curServicePos

private java.lang.String **curServicePos**

curServiceClass

private java.lang.String **curServiceClass**

curUserCountryId

private java.lang.String **curUserCountryId**

curUserId

private int **curUserId**

dbe

public java.lang.Exception **dbe**

user

public dbsection.UserBean **user**

avaFiles

public java.util.Vector **avaFiles**

enum

private java.util Enumeration **enum**

serverName

private java.lang.String **serverName**

serverPass

private java.lang.String **serverPass**

filename

private java.lang.String **filename**

server

private java.lang.String **server**

savedFilename

private java.lang.String[] **savedFilename**

Constructor Detail

ServerLogin

public **ServerLogin()**

Method Detail

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest request,  
                   javax.servlet.http.HttpServletResponse response)  
    throws javax.servlet.ServletException,  
           java.io.IOException
```

Processing user's POST request by simply passing the control to the doGet.

Overrides:

doPost in class javax.servlet.http.HttpServlet

Parameters:

request - The client's request.

response - The response to the client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,  
                  javax.servlet.http.HttpServletResponse response)  
    throws javax.servlet.ServletException,  
           java.io.IOException
```

Processing user's GET request.

Overrides:

doGet in class javax.servlet.http.HttpServlet

Parameters:

request - the client request.

response - the response to client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

validateServer

```
private void validateServer(java.lang.String name,  
                             java.lang.String pass)
```

Authenticates a server with the given username & password

Parameters:

name - The server's username.

pass - The server's password.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a servlet error encountered

findVersion

private java.lang.String[] **findVersion**(java.lang.String fullName)
This method analyses the filename to extract the date and the time, the file saved in the system which actually represents the version of the file.

Parameters:

fullName - The complete name of the file

Returns:

An array of string containing the date and the time

findFiles

private java.util.Vector **findFiles**(java.lang.String serverCode)
Finds all the files that are releasable to that specific server according to the security authorizations contained in the database

Parameters:

serverCode - The code of this server that recognized by the system

Returns:

A Vector containing the releasable files

Throws:

java.lang.Exception - If an error found during the process

openConnection

private void **openConnection**()

Opens a connection with the database to retrieve the available files of this user

Returns:

void

dbsection
Class UpdateDb

```

java.lang.Object
|
+-javax.servlet.GenericServlet
|
+-javax.servlet.http.HttpServlet
|
+-dbsection.UpdateDb

```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

public class **UpdateDb**

extends javax.servlet.http.HttpServlet

The UpdateDb class gets an already parsed file, updates the database and saves the file in the "secure" server area.

See Also:

[Serialized Form](#)

Field Summary	
private dbsection.DBConnectionBean	<u>dbBean</u>
private java.lang.String	<u>desiredClass</u>
private int	<u>fieldId</u>
private java.lang.String	<u>filename</u>
private java.lang.String	<u>finalFilename</u>
private dbsection.UserBean	<u>newuser</u>
private java.io.PrintWriter	<u>out</u>
private javax.servlet.http.HttpServletRequest	<u>req</u>

private javax.servlet.http.HttpServletRespons e	res
private java.lang.String	tempFile

Fields inherited from class javax.servlet.http.HttpServlet

Fields inherited from class javax.servlet.GenericServlet

Constructor Summary
UpdateDb()

Method Summary	
void private	copyFile() Copies the file from the temporary server area to the simulated permanent secure area
void private	createFinalFilename() Creates the final filename that the file will get in order to be saved, by adding the date and the time
private java.lang.String	createUserDirectory() Creates a new directory according to user's classification in case there is not one yet
void	doGet() (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Processing user's GET request by simply passing the control to the doPost.
void	doPost() (javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res) Processing user's POST request.
void private	exitLinks() Displays the available links to direct the control on another servlet, in case the user wants to cancel this procedure
void private	openConnection() Opens a connection with the database to retrieve the available files of this user

private void	<u>printData()</u> Prints the user's personal and Security data retrieved from the database that contained in the UserBean object
private boolean	<u>updateDatabase()</u> Updates the database with the new file's data and metadata

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

newuser

private dbsection.UserBean **newuser**

out

private java.io.PrintWriter **out**

req

private javax.servlet.http.HttpServletRequest **req**

res

private javax.servlet.http.HttpServletResponse **res**

fieldId

private int **fieldId**

desiredClass

private java.lang.String **desiredClass**

tempFile

```
private java.lang.String tempFile
```

```
filename
```

```
private java.lang.String filename
```

```
finalFilename
```

```
private java.lang.String finalFilename
```

```
dbBean
```

```
private dbsection.DBConnectionBean dbBean
```

Constructor Detail

```
UpdateDb
```

```
public UpdateDb()
```

Method Detail

```
doGet
```

```
public void doGet(javax.servlet.http.HttpServletRequest request,  
                  javax.servlet.http.HttpServletResponse response)  
    throws javax.servlet.ServletException,  
           java.io.IOException
```

Processing user's GET request by simply passing the control to the doPost.

Overrides:

doGet in class javax.servlet.http.HttpServlet

Parameters:

request - The client request.

response - The response to client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

```
doPost
```

```
public void doPost(javax.servlet.http.HttpServletRequest req,  
                   javax.servlet.http.HttpServletResponse res)  
    throws java.io.IOException
```

Processing user's POST request.

Overrides:

doPost in class javax.servlet.http.HttpServlet

Returns:

void.

Throws:

`javax.servlet.ServletException` - In case of a servlet error encountered

`java.io.IOException` - In case of a I/O error encountered

updateDatabase

private boolean **updateDatabase()**

Updates the database with the new file's data and metadata

Returns:

A boolean variable is returned to verify the success of the procedure

copyFile

private void **copyFile()**

throws `java.io.IOException`

Copies the file from the temporary server area to the simulated permanent secure

area

Returns:

void.

Throws:

`java.io.IOException` - In case of a I/O error encountered

createFinalFilename

private void **createFinalFilename()**

Creates the final filename that the file will get in order to be saved, by adding the date and the time

Returns:

void.

createUserDirectory

private `java.lang.String` **createUserDirectory()**

throws `java.lang.SecurityException`

Creates a new directory according to user's classification in case there is not one

yet

Returns:

The newly created directory

Throws:

`java.lang.SecurityException` - In the case of the creation is not allowed by the

security manager

openConnection

private void **openConnection()**

Opens a connection with the database to retrieve the available files of this user

Returns:

void

printData

private void **printData()**

Prints the user's personal and Security data retrieved from the database that contained in the UserBean object

Returns:

void.

exitLinks

private void **exitLinks()**

Displays the available links to direct the control in another servlet, in case the user wants to cancel this procedure

Returns:

void.

dbsection
Class UserBean

```
java.lang.Object
|
+--dbsection.UserBean
```

```
public class UserBean
extends java.lang.Object
This class defines a bean used to maintain user's data during the session.
```

Field Summary	
<pre>private java.util.Vecto r</pre>	<u>avaFiles</u>
<pre>private java.lang.Strin g</pre>	<u>city</u>
<pre>private java.lang.Strin g</pre>	<u>countryId</u>
<pre>private java.lang.Strin g</pre>	<u>email</u>
<pre>private java.lang.Strin g</pre>	<u>firstName</u>
<pre>private java.lang.Strin g</pre>	<u>lastName</u>
<pre>private java.lang.Strin g</pre>	<u>loginName</u>
<pre>private java.lang.Strin g</pre>	<u>middleInitial</u>
<pre>private java.lang.Strin g</pre>	<u>password</u>
<pre>private java.lang.Strin g</pre>	<u>phone</u>

private java.lang.String	<u>serviceClass</u>
private java.lang.String	<u>serviceName</u>
private java.lang.String	<u>servicePos</u>
private java.lang.String	<u>state</u>
private java.lang.String	<u>street</u>
private int	<u>userID</u>
private java.lang.String	<u>zip</u>

Constructor Summary

[UserBean](#) ()

The default constructor initializes a new object by calling the getReset() method

Method Summary

java.util.Vector	<u>getAvaFiles</u> () Gets user's available files stored in the object
java.lang.String	<u>getCity</u> () Gets user's city name
java.lang.String	<u>getCountryId</u> () Gets user's country ID
java.lang.String	<u>getEmail</u> () Gets user's email address
java.lang.String	<u>getFirstName</u> () Gets user's first name
java.lang.String	<u>getFullName</u> () Gets user's full name name (first + middle + last)
java.lang.String	<u>getLastName</u> () Gets user's last name
java.lang.String	<u>getLoginName</u> () Gets user's login name

g	java.lang.String	<u>getMiddleInitial</u> () Gets user's middle initial
g	java.lang.String	<u>getPassword</u> () Gets user's password
g	java.lang.String	<u>getPhone</u> () Gets user's phone number
g	java.lang.String	<u>getReset</u> () Resets all fields by creating new empty objects
g	java.lang.String	<u>getServiceClass</u> () Gets user's service classification
g	java.lang.String	<u>getServiceName</u> () Gets user's service name
g	java.lang.String	<u>getServicePos</u> () Gets user's service position
g	java.lang.String	<u>getState</u> () Gets user's state
g	java.lang.String	<u>getStreet</u> () Gets user's street
	int	<u>getUserID</u> () Gets user's ID
g	java.lang.String	<u>getZip</u> () Gets user's zip
g	java.lang.String	<u>secString</u> () Gets user's security data all in a String
	void	<u>setAvaFiles</u> (java.util.Vector newAvaFiles) Sets user's available files with a new value
	void	<u>setCity</u> (java.lang.String newCity) Sets user's city name with a new value
	void	<u>setCountryId</u> (java.lang.String newCountryId) Sets user's country ID with a new value
	void	<u>setEmail</u> (java.lang.String newEmail) Sets user's email address with a new value
	void	<u>setFirstName</u> (java.lang.String newFirstName) Sets user's first name with a new value
	void	<u>setLastName</u> (java.lang.String newLastName) Sets user's last name with a new value
	void	<u>setLoginName</u> (java.lang.String newLoginName) Sets user's login name with a new value

void	<u>setMiddleInitial</u> (java.lang.String newMiddleInitial) Sets user's middle initial with a new value
void	<u>setPassword</u> (java.lang.String newPassword) Sets user's password with a new value
void	<u>setPhone</u> (java.lang.String newPhone) Sets user's phone number with a new value
void	<u>setServiceClass</u> (java.lang.String newServiceClass) Sets user's service classification with a new value
void	<u>setServiceName</u> (java.lang.String newServiceName) Sets user's service name with a new value
void	<u>setServicePos</u> (java.lang.String newServicePos) Sets user's service position with a new value
void	<u>setState</u> (java.lang.String newState) Sets user's state with a new value
void	<u>setStreet</u> (java.lang.String newStreet) Sets user's street name with a new value
void	<u>setUserID</u> (int newUserID) Sets user ID with a new value
void	<u>setZip</u> (java.lang.String newZip) Sets user's zip code with a new value
java.lang.String	<u>toString</u> () Gets user's personal data all in a String

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

userID

private int **userID**

firstName

private java.lang.String **firstName**

middleInitial

private java.lang.String **middleInitial**

lastName

private java.lang.String **lastName**

street

private java.lang.String **street**

city

private java.lang.String **city**

zip

private java.lang.String **zip**

state

private java.lang.String **state**

phone

private java.lang.String **phone**

loginName

private java.lang.String **loginName**

password

private java.lang.String **password**

email

private java.lang.String **email**

serviceName

private java.lang.String **serviceName**

servicePos

private java.lang.String **servicePos**

serviceClass

private java.lang.String **serviceClass**

countryId

private java.lang.String **countryId**

avaFiles

private java.util.Vector **avaFiles**

Constructor Detail

UserBean

public **UserBean()**

The default constructor initializes a new object by calling the `getReset()` method

Method Detail

getReset

public java.lang.String **getReset()**

Resets all fields by creating new empty objects

Returns:

An empty String

getUserID

public int **getUserID()**

Gets user's ID

Returns:

the user's ID

setUserID

public void **setUserID**(int newUserID)

Sets user ID with a new value

Parameters:

`newUserID` - the new userID

Returns:

void

getFirstName

public java.lang.String **getFirstName()**

Gets user's first name

Returns:

the user's first name

setFirstName

public void **setFirstName**(java.lang.String newFirstName)

Sets user's first name with a new value

Parameters:

newFirstName - The new first name

Returns:

void

getMiddleInitial

public java.lang.String **getMiddleInitial**()

Gets user's middle initial

Returns:

the user's middle initial

setMiddleInitial

public void **setMiddleInitial**(java.lang.String newMiddleInitial)

Sets user's middle initial with a new value

Returns:

void

getLastName

public java.lang.String **getLastName**()

Gets user's last name

Returns:

the user's last name

setLastName

public void **setLastName**(java.lang.String newLastName)

Sets user's last name with a new value

Returns:

void

getFullName

public java.lang.String **getFullName**()

Gets user's full name name (first + middle + last)

Returns:

the user's full name

setStreet

public void **setStreet**(java.lang.String newStreet)
Sets user's street name with a new value
Returns:
void

setCity

public void **setCity**(java.lang.String newCity)
Sets user's city name with a new value
Returns:
void

setZip

public void **setZip**(java.lang.String newZip)
Sets user's zip code with a new value
Returns:
void

setState

public void **setState**(java.lang.String newState)
Sets user's state with a new value
Returns:
void

setPhone

public void **setPhone**(java.lang.String newPhone)
Sets user's phone number with a new value
Returns:
void

setLoginName

public void **setLoginName**(java.lang.String newLoginName)
Sets user's login name with a new value
Returns:
void

setPassword

public void **setPassword**(java.lang.String newPassword)
Sets user's password with a new value
Returns:
void

setEmail

public void **setEmail**(java.lang.String newEmail)
Sets user's email address with a new value
Returns:
void

setServiceName

public void **setServiceName**(java.lang.String newServiceName)
Sets user's service name with a new value
Returns:
void

setServicePos

public void **setServicePos**(java.lang.String newServicePos)
Sets user's service position with a new value
Returns:
void

setServiceClass

public void **setServiceClass**(java.lang.String newServiceClass)
Sets user's service classification with a new value
Returns:
void

setCountryId

public void **setCountryId**(java.lang.String newCountryId)
Sets user's country ID with a new value
Returns:
void

setAvaFiles

public void **setAvaFiles**(java.util.Vector newAvaFiles)
Sets user's available files with a new value
Returns:
void

getStreet

public java.lang.String **getStreet**()

Gets user's street

Returns:

the user's street

getCity

public java.lang.String **getCity()**

Gets user's city name

Returns:

the user's city name

getZip

public java.lang.String **getZip()**

Gets user's zip

Returns:

the user's zip

getState

public java.lang.String **getState()**

Gets user's state

Returns:

the user's state

getPhone

public java.lang.String **getPhone ()**

Gets user's phone number

Returns:

the user's phone number

getLoginName

public java.lang.String **getLoginName()**

Gets user's login name

Returns:

the user's login name

getPassword

public java.lang.String **getPassword()**

Gets user's password

Returns:

the user's password

getEmail

public java.lang.String **getEmail()**

Gets user's email address

Returns:

the user's email address

getServiceName

public java.lang.String **getServiceName()**

Gets user's service name

Returns:

the user's service name

getServicePos

public java.lang.String **getServicePos()**

Gets user's service position

Returns:

the user's service position

getServiceClass

public java.lang.String **getServiceClass()**

Gets user's service classification

Returns:

the user's service classification

getCountryId

public java.lang.String **getCountryId()**

Gets user's country ID

Returns:

the user's country ID

getAvaFiles

public java.util.Vector **getAvaFiles()**

Gets user's available files stored in the object

Returns:

the user's available files

toString

public java.lang.String **toString()**

Gets user's personal data all in a String

Overrides:

toString in class java.lang.Object

Returns:

the user's personal data

secString

public java.lang.String **secString()**

Gets user's security data all in a String

Returns:

the user's security data

dbsection

Class **UserOptions**

java.lang.Object

|

+javax.servlet.GenericServlet

|

+javax.servlet.http.HttpServlet

|

+**dbsection.UserOptions**

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

public class **UserOptions**

extends javax.servlet.http.HttpServlet

The class UserOptions creates a page displaying the user's information (personal and security) and the available options for that session. After the user's selection pass the control to the next UserSelection servlet

See Also:

[Serialized Form](#)

Field Summary

private java.util.Vector	avaFiles
private dbsection.UserBean	newuser
private java.io.PrintWriter	out

Fields inherited from class javax.servlet.http.HttpServlet

Fields inherited from class javax.servlet.GenericServlet

Constructor Summary	
UserOptions ()	

Method Summary	
void	doGet (javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res) Processing user's GET request.
private void	printData () Prints the user's personal and Security data retrieved from the database that contained in the UserBean object
private void	printOptions () Creates a drop down list that presents the user's available options.

Methods inherited from class javax.servlet.http.HttpServlet
doDelete, doOptions, doPost, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet
destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail
newuser
private dbsection.UserBean newuser
avaFiles
private java.util.Vector avaFiles
out
private java.io.PrintWriter out
Constructor Detail
UserOptions

public **UserOptions()**

Method Detail

doGet

public void **doGet**(javax.servlet.http.HttpServletRequest req,
 javax.servlet.http.HttpServletResponse res)
 throws java.io.IOException

Processing user's GET request.

Overrides:

doGet in class javax.servlet.http.HttpServlet

Parameters:

req - the client request.

res - the response to client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

printData

private void **printData()**

Prints the user's personal and Security data retrieved from the database that
contained in the UserBean object

Returns:

void.

printOptions

private void **printOptions()**

Creates a drop down list that presents the user's available options.

Returns:

void.

dbsection
Class UserSelection

```
java.lang.Object
|
+-javax.servlet.GenericServlet
|
+-javax.servlet.http.HttpServlet
|
+-dbsection.UserSelection
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

public class **UserSelection**

extends javax.servlet.http.HttpServlet

The UserSelection class is responsible to get the user's selection from the main menu and to redirect the control to the respective servlet for further processing

See Also:

[Serialized Form](#)

Field Summary	
private java.util.Vector	avaFiles
private java.lang.String	FileMenu
private dbsection.User bean	freshuser
private java.lang.String	nextServlet
private java.io.PrintWriter	out
private javax.servlet.http.HttpServletRequest	req
private javax.servlet.http.HttpServletResponse	res

Fields inherited from class javax.servlet.http.HttpServlet

Fields inherited from class javax.servlet.GenericServlet

Constructor Summary

[UserSelection](#) ()

Method Summary

private void	continueSession () Continue the session and pass the control to the next servlet.
void	doGet (javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response) Processing user's GET request by simply passing the control to the doPost.
void	doPost (javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res) Processing user's POST request.

Methods inherited from class javax.servlet.http.HttpServlet

doDelete, doOptions, doPut, doTrace, getLastModified, service, service

Methods inherited from class javax.servlet.GenericServlet

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

req

private javax.servlet.http.HttpServletRequest req

res

private javax.servlet.http.HttpServletResponse **res**

freshuser

private dbsection.UserBean **freshuser**

avaFiles

private java.util.Vector **avaFiles**

out

private java.io.PrintWriter **out**

nextServlet

private java.lang.String **nextServlet**

FileMenu

private java.lang.String **FileMenu**

Constructor Detail

UserSelection

public **UserSelection()**

Method Detail

doGet

public void **doGet**(javax.servlet.http.HttpServletRequest request,
 javax.servlet.http.HttpServletResponse response)
 throws javax.servlet.ServletException,
 java.io.IOException

Processing user's GET request by simply passing the control to the doPost.

Overrides:

doGet in class javax.servlet.http.HttpServlet

Parameters:

request - The client's request.

response - The response to the client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,  
                  javax.servlet.http.HttpServletResponse res)  
    throws java.io.IOException
```

Processing user's POST request.

Overrides:

doPost in class javax.servlet.http.HttpServlet

Parameters:

req - the client request.

res - the response to client.

Returns:

void.

Throws:

javax.servlet.ServletException - In case of a servlet error encountered

java.io.IOException - In case of a I/O error encountered

continueSession

```
private void continueSession()  
    throws java.lang.Exception
```

Continue the session and pass the control to the next servlet.

Returns:

void.

Throws:

Exception. - In case an invalid situation is encountered

java.lang.Exception

APPENDIX B. JAVA CODE

```

/*

*****
* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
*   Date             :      December 2002
*****

*/

package dbsection;

import java.io.*;
import java.util.*;
import java.sql.*;

import dbsection.*;

/**
 * This class implements a bean that is used from many other classes
 * to process all transactions with the backend database.
 * It is a unique point of connection with the database,
 * using the JDBC API.
 * @author Panos
 */
public class DBConnectionBean
{
    private boolean loaded;
    private boolean connected;
    private String url;
    private String driver;
    private String user;
    private String password;
    private Connection con;
    private Statement statement;
    private String query;
    private String update;
    private String primaryKeyQuery;

    /**
     * The default constructor of the class that sets the
     * appropriate url and driver for the database
     * @return void.
     */
    public void DBConnectionBean()
    {
        loaded = false;
        connected = false;

        url = "jdbc:odbc:UserDB" ;

```

```

driver = "sun.jdbc.odbc .JdbcOdbcDriver" ;
user = "panos" ;
password = "greek" ;

con = null;
query = new String("");
primaryKeyQuery = new String();
}

/**
 * Set the URL of the database.
 * @param newUrl The new URL
 * @return void.
 */
public void setUrl(String newUrl)
{
    url = newUrl;
}

/**
 * Sets the driver for the database
 * @param newDriver The new driver.
 * @return void.
 */
public void setDriver(String newDriver)
{
    driver = newDriver;
}

/**
 * Sets the DB user name.
 * @param newUser The new user name.
 * @return void.
 */
public void setUser(String newUser)
{
    user = newUser;
}

/**
 * Sets the DB password.
 * @param newPassword The new password.
 * @return void.
 */
public void setPassword(String newPassword)
{
    password = newPassword;
}

/**
 * Loads the driver

```

```

    * @return True if the driver is loaded successfully
    */
    public boolean isLoaded()
    {
        try
        {
            Class.forName(driver);
            loaded = true;
        }
        catch(ClassNotFoundException cnfe)
        {
            loaded = false;
        }
        return loaded;
    }

    // end of isLoaded()

    /**
     * Establish the connection with the DB.
     * @return True if the connection is established.
     */
    private boolean isConnected()
    {
        try
        {
            con = DriverManager.getConnection(url, user, password);
            statement = con.createStatement();
            connected = true;
        }
        catch(SQLException sqle) {
            connected = false;
        }
        return connected;
    }

    /**
     * Sets the query string.
     * @param query The new query string.
     * @return A ResultSet object containing the results
     *         of the query
     */
    public ResultSet query(String query)
    {
        this.query = query;
        return getQuery();
    }

    /**
     * Queries the DB and return the result ResultSet object
     * @return A ResultSet object containing the results
     *         of the query

```

```

    */
    public ResultSet getQuery()
    {
        ResultSet rs = null;
        if(isConnected())
        {
            try
            {
                rs = statement.executeQuery(query);
            }
            catch(Exception e)
            {
                System.out.println("Inside DBConnection exception is = " + e);
            }
        }
        return rs;
    }

    /**
     * Sets the query.
     * @param query The new query.
     * @return void.
     */
    public void setQuery(String query)
    {
        this.query = query;
    }

    /**
     * Sets the update query.
     * @param update The update query.
     * @return void.
     */
    public void setUpdate(String update)
    {
        this.update = update;
    }

    /**
     * Executes an update query by calling the getUpdate method
     * @param updateQuery the update query
     * @return An integer representing the primary key
     *         or -1 if an error happens.
     */
    public int update(String updateQuery )
    {
        update = updateQuery;
        return getUpdate();
    }

```

```

/**
 * Executes an update query.
 * @return An integer representing the primary key
 *         or -1 if an error happens.
 */
public int getUpdate()
{
    int result = -1;
    if(isConnected())
    {
        try
        {

            result = statement.executeUpdate(update);
            isClose();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    return result;
}

/**
 * This method closes the connection with the database
 * @return A boolean true if the connection is closed
 *         successfully
 */
public boolean isClose()
{
    try
    {
        {
            con.close();
            statement.close();
        }
        catch(Exception e) {}
        connected = false;

        return true;
    }
}

/**
 * Sets the primary key for the query.
 * @param newPrimaryKeyQuery The new primary key for that query
 * @return void.
 */
public void setPrimaryKeyQuery(String newPrimaryKeyQuery)
{
    primaryKeyQuery = newPrimaryKeyQuery;
}

```

```

/**
 * Obtains a new primary key in the table specified by the current primary.
 * The result set of the ordered set of existing primary keys is examined
 * sequentially, until the smallest non used positive integer is found.
 * This method may only be used for tables using integers as a primary key.
 * @return the new primary key or -1 if an error happens.
 */
public int getPrimaryKey()
{
    int primaryKey = -1;
    try
    {
        ResultSet rs = query(primaryKeyQuery);
        if(rs.next())
        {
            primaryKey = rs.getInt(1);
            do
            {
                ++primaryKey;
            } while(rs.next() && (primaryKey == rs.getInt(1)));
        }
    }
    catch(SQLException e)
    {
        e.printStackTrace();
        primaryKey = -1;
    }
    return primaryKey;
}

} // End of DBConnectionBean class

```

```

/*
*****
* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
* Date               :      December 2002
*****

*/
package dbsection;

import java.io.*;
import java.io.Writer ;
import java.net.* ;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

/**
 * The class Echo24 is responsible to
 * parse the passed file and extract all of its
 * elements and attributes
 * @author Panos
 */
public class Echo24 extends DefaultHandler
{
    // Data member
    static private Writer out;
    private String indentString = "  "; // Amount to indent
    private int indentLevel = 0;

    private String[] parsedFile ;
    private File      file ;
    private FileTags  fileTags ;

    private boolean secElementsFound = false ;
    private boolean usSecElementFound = false ;
    private boolean nonusSecElementFounf = false ;
    private String  secCategory ="" ;
    private int     index=0 ;

    /**
     * The default constructor of the class
     */
    public Echo24()
    {

```

```

    }

/**
 * This method parses the file by using one
 * of the SAXParsers from the SAXParserFactory
 * @param filename The name of the file to be parsed
 * @return A FileTags object containing all the security tags
 *         of the passed file
 * @exception Throwable In case of an error during the process
 */
public FileTags parseFile(String filename)
{
    // Use an instance of ourselves as the SAX event handler *****
    DefaultHandler handler = Echo24.this ;

    // Use the default parser
    SAXParserFactory factory = SAXParserFactory.newInstance();

    try
    {
        // Set up output stream
        out = new OutputStreamWriter(System.out, "UTF8");
        file = new File(filename) ;

        // pass the file to the new fileTags object
        // so this fileTags object is related to this file
        fileTags = new FileTags(file) ;

        // Parse the input
        SAXParser saxParser = factory.newSAXParser();
        saxParser.parse(filename , handler);

        parsedFile = this.getParsedFile() ;

    }
    catch (Throwable t)
    {
        fileTags=null;
    }

    return fileTags ;
}

/**
 * This method parses the file in the passed URL by using one
 * of the SAXParsers from the SAXParserFactory
 * @param filename The name of the file to be parsed
 * @return A FileTags object containing all the security tags
 *         of the passed file

```

```

* @exception Throwable In case of an error during the process
*/
public FileTags mainNew(String argv)
{
    // Use an instance of ourselves as the SAX event handler *****
    DefaultHandler handler = Echo24.this ;

    // Use the default parser
    SAXParserFactory factory = SAXParserFactory.newInstance();

    try
    {
        URL checkUrl = new URL(argv) ;

        // Set up output stream
        out = new OutputStreamWriter(System.out, "UTF8");
        file = new File(argv) ;

        // pass the file to the new fileTags object
        // so this fileTags object is related to this file
        fileTags = new FileTags(file) ;

        // Parse the input
        SAXParser saxParser = factory.newSAXParser();

        saxParser.parse( argv , handler);

        parsedFile = this.getParsedFile() ;

    }
    catch (Throwable t)
    {
        t.printStackTrace();
    }

    return fileTags ;
}

/**
 * Gets the file contained in the object
 * @return The file contained in the object
 */
public File getFile()
{
    return file ;
}

/**
 * Gets the parsed file contained in the object
 * @return The file parsed contained in the object

```

```

*    in an array of String
*/
    public String[] getParsedFile()
    {
        return parsedFile ;
    }

    public void startElement(String namespaceURI,
        String lName, // local name
        String qName, // qualified name
        Attributes attrs)
        throws SAXException
    {
        indentLevel++;

        String eName = lName; // element name

        if ("".equals(eName)) eName = qName; // namespaceAware = false

        // means that there are some security tags
        if(eName.equals("Security")    ) secElementsFound =true ;

        if(secElementsFound)
        {

            boolean foundSecElement = fileTags.checkElement(secCategory, eName , "YES IT IS")
;

            if (attrs != null)
            {
                for (int i = 0; i < attrs.getLength(); i++)
                {
                    String aName = attrs.getLocalName(i); // Attr name
                    if ("".equals(aName)) aName = attrs.getQName(i);

                    fileTags.checkAttribute(secCategory, eName, aName, attrs.getValue(i) ) ;

                }

            } // if attrs!=null

            if (attrs.getLength() > 0) nl();

        }

    } // startElement

    public void endElement(String namespaceURI,

```

```

        String sName, // simple name
        String qName // qualified name
    )
    throws SAXException
    {
        if( (qName.equals("Security"))) secElementsFound = false ;

        indentLevel--;
    }

// *****
public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        String s = new String(buf, offset, len);
        if (!s.trim().equals("")) emit(s);
    }

// Wrap I/O exceptions in SAX exceptions, to
// suit handler signature requirements
private void emit(String s)
    throws SAXException
    {

    }

// Start a new line
// and indent the next line appropriately
private void nl()
    throws SAXException
    {
        String lineEnd = System.getProperty("line.separator");
        try {
            out.write(lineEnd);
            for (int i=0; i < indentLevel; i++) out.write(indentString);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }
}

```

```

/*
*****
* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
*   Date             :      December 2002
*****

*/
package dbsection;

import dbsection.* ;
import java.io.*;
import java.io.Writer ;

/**
 * The class FileTags is responsible to
 * create an object containing all the security
 * elements and attributes of the specific file
 * that belongs
 * @author Panos
 */

public class FileTags
{
    // Data Members
    String filename ;
    String[] secElement={ "CIA-IUO" , "CLASSGUIDE" ,
                          "COMSEC" , "COUNTRY-
ID", "NONUS-SEC", "NOFORN"
    };

    String[] secElementValue={ "empty" , "empty" , "empty" ,
                                "empty" ,
                                "empty" };

    String[] secAttribute={ "classification" , "nonUSmarkings" ,
                             "FGIsourceOpen" , "SCIcontrols" ,
                             "disseminationControls" , "FGIsourceProtected" ,
                             "nonICmarkings" } ;

    String[] secAttributeValue={ "empty" , "empty" , "empty" ,

```

```

"empty" ,
"empty" } ;

```

```

File file ;

```

```

/**
 * The default constructor of the class
 * receives the passed File object and initializes
 * the parameter for itself
 * @param newFile The File object on which that FileTags
 * object is referred to
 */
public FileTags(File newFile)
{
    file = newFile ;
}

/**
 * Sets the name of the file
 * @param newFileName The new filename
 * @return void
 */
private void setFileName(String newFileName)
{
    filename = newFileName ;
}

/**
 * Gets file's security attributes contained in the object
 * @return The file's security attributes contained in the object
 */
public String[] getSecAttributes()
{
    return secAttribute ;
}

/**
 * Gets file's security attributes values contained in the object
 * @return The file's security attributes values contained in the object
 */
public String[] getSecAttributeValue()
{
    return secAttributeValue ;
}

/**
 * Checks the passed element if it is a
 * security element
 * @param category The element's category

```

```

* @param newElement The element to be checked
* @param newElementValue The element's value
* @return A boolean true if the element is a security element
*/
public boolean checkElement(String category,
                           String newElement ,
                           String newElementValue)
{
    boolean hasSecElements = false ;

    // check all the security elements to find if the newTag
    // is a security element
    for (int i=0; i< secElement.length ; i++)
    {
        if ( secElement[i].equals(newElement) )
        {
            secElementValue[i]= newElementValue ;
            hasSecElements = true ;
        }
    }

    return hasSecElements ;
}

/**
* Checks the passed attribute if it is a
* security attribute
* @param category The attribute's category
* @param newElement The element containing the attribute
* @param newAttribute The attribute to be checked
* @param newAttributeValue The attribute's value
* @return void
*/
public void checkAttribute( String category,
                           String newElement ,
                           String newAttribute ,
                           String newAttributeValue)
{
    boolean hasSecAttributes = false ;

    // check all the security attributes to find if the newAttribute
    // is a security element
    for (int i=0; i< secAttribute.length ; i++)
    {
        if ( secAttribute[i].equals(newAttribute) )
        {
            secAttributeValue[i]= newAttributeValue ;
            hasSecAttributes = true ;

            i=(secAttribute.length - 1) ;
        }
    }
}

```

```
        }  
    }  
}  
    //FileTags class ends here
```

```

/*

*****

* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
* Date               :      December 2002

*****

*/

package dbsection;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.net.*;
import java.util.*;
import javax.activation.*;

import dbsection.*;

/**
 * The FindUrl class defines a servlet that is responsible to connect
 * the system with the requested server, to retrieve the
 * releasable files from that server and to decide which
 * of those files the user is allowed to access
 */
public class FindUrl extends HttpServlet implements Serializable
{

    // Data Members
    private UserBean    user ;
    private      PrintWriter out ;
    private  HttpServletRequest req;
    private  HttpServletResponse res;

    private      String selUrl;
    private String selectedUrl="nothingYet" ;
    private String permission="noneYet" ;
    private String authentication="noneYet" ;

    private static  FileTags fileTags ;
    private static  SecMetadataManager manager ;
    private boolean decision=false ;

    private File file ;
    // thisServer is kind of the login name of the server
    private String thisServer = "http://localhost:8080" ;
    private String thisServerPassword = "password" ;
    private Vector serverFoundFiles ;

```

```

        private Vector matchedFiles ;

        private          String npsServer = "http://131.120.8.193:8080/metadata/servlet/"
;
        private          String localServer = "http://localhost:8080/metadata/servlet/" ;
        private          String server ;
        private          String ReqFilename ;

/**
 * Processing user's GET request by simply passing
 * the control to the doPost.
 * @param request The client's request.
 * @param response The response to the client.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a I/O error encountered
 */
public void doGet(Http ServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
{
    this.doPost(request, response);
    return;
} // end of doGet

/**
 * Processing user's POST request.
 * @param req the client request.
 * @param res the response to client.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a I/O error encountered
 */
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws IOException
{
    // Initialization
    this.req = req;
    this.res = res;

    serverFoundFiles = new Vector() ;

    // get the session object from the previous servlet
    HttpSession session = req.getSession(true);

    // get now the UserBean object from the session
    user = (UserBean) session.getAttribute("user");

    // Get the user selections / server and filename
    server          = req.getParameter("server") ;
    selectedUrl = req.getParameter("fileUrl");

```

```

// "nothing" means that the request is coming from the other server
if(selectedUrl.equals("nothing"))
{
    // checks the authentication if it was succesfull
    authentication = req.getParameter("permission");
    ReqFilename     = req.getParameter("filename");
}

// Sets the content type to html text
res.setContentType("text/html");

// Gets the PrintWriter object for sending HTML commands
out = res.getWriter();

//Generates the title
out.println("<html><head><title>");
out.println("Find the requested (URL) file servlet");
out.println("</title></head>");

//Generates the body
out.println("<body bgcolor=\"#fafca3\">");

    // Print the Personal and the security data
    printData();

    // connection established the available files presented
    if(authentication.equals("approved"))
    {
out.println(" The total available files on the server " + server + "<br>\n");

        for (int i=0; i<user.getAvaFiles().size(); i++)
        {
out.println(" ***** " + user.getAvaFiles().elementAt(i) + "<br>\n");
        }

        // find those files that have a matching filename
        matchedFiles = new Vector();
        matchedFiles = checkIfExist(ReqFilename);

        user.setAvaFiles(matchedFiles);

        String fileClass;
        String file;
        out.println(" Files containing " + ReqFilename + " /// user's classification is = " +
user.getServiceClass()+ "<br>\n");
        for (int i=0; i<matchedFiles.size(); i++)
        {
            file = (String) matchedFiles.elementAt(i);
            fileClass= file.substring(0,2);

```

```

        // find the releasable files according to user's classification
        if( fileClass.equals(user.getServiceClass() ) )
        {
            out.println(" ----- " + matchedFiles.elementAt(i) + " ---- file class = " +
fileClass+ "<br>\n");
        }
    }

    String flag = "server" ;
    out.println("<a href=\"" + res.encodeURL("Op enFile?selFile=" + flag )
        + ">" + " continue to open the file you want " +
"</a>" + "<br>\n");
}

// initial stage. Trying to connect
if(!selectedUrl.equals("nothing"))
{
    System.out.println(" inside inside initial Stage , server=" +server );
    System.out.println(" inside inside initial Stage , selectedUrl =" + selectedUrl );

    if(server.equals("NPS")) server=npsServer ;
    if(server.equals("LOCAL")) server=localServer ;
    if(server.equals("OTHER")) server=selectedUrl ;

    System.out.println(" inside inside initial Stage , after if server=" +server );

    getConnected(server) ;
}

exitLinks(); // Create and print the links to other servlets

out.println("</body></html>");
} // doPost ends here

/**
 * Starts a new connection with the associated server
 * @param server The associated server to be connected
 * @return void.
 * @exception Exception.
 */
private void getConnected(String server)
{
    try
    {
        HttpSession session = req.getSession(true);
        if(!session.isNew())
        {
            // Ensure the session is newly created
            session.invalidate();
            session = req.getSession(true);

```

```

        }

        // Attach customer bean to this session object
        session.setAttribute("user", user);

        res.sendRedirect( res.encodeRedirectURL( server
            + "dbsection.ServerLogin?serverID="
            + thisServer + "&password="
            + thisServerPassword + "&filename="
            + selectedUrl + "&server=" + server) )
;

        }
        catch (Exception e)
        {
            // openConnection() failed
            System.out.println(" IOException ..... " + e );
        }
    }

/**
 * Starts a new session after a customer is correctly authenticated.
 * @return void.
 * @exception Exception.
 */
private void startSession() throws Exception
{
    // The next servlet is called
    // Get the dispatcher
    RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher( server
            + "dbsection.ServerLogin?serverID="
            + thisServer + "&password="
            + thisServerPassword + "&filename="
            + selectedUrl + "&server=" + server);

    if (dispatcher == null)
    {
        // No dispatcher means the given file could not be found
        // response.sendError(response.SC_NO_CONTENT);
    }
    else
    {
        // Get or start a new session for this user
        HttpSession session = req.getSession(true);
        if(!session.isNew())
        {
            // Ensure the session is newly created
            session.invalidate();

```

```

        session = req.getSession(true);
    }

    // Attach customer bean to this session object
    session.setAttribute("user", user);

    // Pass control to a different page
    dispatcher.forward(req, res);
}

} // end of startSession()

// *****
/**
 * Checks the passed filename if exist within the name of the files
 * @return A Vector containing the found files.
 * @exception Exception.
 */

private Vector checkIfExist(String fileName)
{
    String possibleFile;
    String testStr;
    Vector matchingFiles = new Vector();
    int widthS = fileName.length();

    Enumeration enum = user.getAvaFiles().elements();
    // check every element of the serverFoundFiles Vector
    while(enum.hasMoreElements())
    {
        possibleFile = (String) enum.nextElement();
        int widthB = possibleFile.length();

        for(int i=0; i<widthB-widthS; i++)
        {
            testStr = possibleFile.substring(i, i+widthS);
            if(testStr.equals(fileName))
            // found
            {
                matchingFiles.add(possibleFile);
            }
            // store the file
            i=widthB-widthS;
            // stop the loop
        }
        // for loop // searching the possibleFile
    }
    // while loop // searching the elements of the Vector

    // add at the end of the Vector the name of the server

```

```

        // who holds those files
        matchingFiles.add(server) ;

        return matchingFiles ;
    }

    private void exitLinks()
    {
        out.println( "<br> <br> <br>\n");

        out.println("<a href=\"" + res.encodeURL("/metadata/html/login.htm")
                    + "\"" + "login again as another user " + "</a>" + " or " +
"<br>\n");
        out.println("<a href=\"" + res.encodeURL("UserOptions")
                    + "\"" + " see again your options" + "</a>" + "<br>\n");

    }

    /**
     * Prints the user's personal and Security data
     * retrieved from the database that contained
     * in the UserBean object
     * @return void.
     */
    private void printData()
    {
        out.println("<h3>" + " Personal Data " + "</h3>");
        out.println("Last Name : " + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getLastName() + "</strong><br>\n" +
"First Name : " + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getFirstName() + "</strong><br>\n");

        out.println("<strong>" + " Security Attributes (presented here only for
demo)" + "</strong><br>\n");
        out.println(" Service Name : " + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getServiceName() + "</strong><br>\n" +
" Service Position:" + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getServicePos() + "</strong><br>\n" +
" Service Classification:" + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getServiceClass() + "</strong><br>\n" +
" Country ID:" + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getCountryId() + "</strong><br>\n"

);
    }

}

} // FindUrl class ends here

```

```

/*

*****

* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
*   Date             :      December 2002

*****

*/

package dbsection;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import javax.activation.*;
import java.awt.*;
import java.*;

/**
 * The ImportFile class defines a servlet that is responsible to
 * create a web page to help user input the file and the server
 * where that file is located in order to open it.
 */
public class ImportFile extends HttpServlet
{

    // Data Members
    private UserBean freshuser ;
    private Vector avaFiles ;
    private PrintWriter out ;
    private HttpServletRequest req;
    private HttpServletResponse res;

    /**
     * Processing user's POST request.
     * @param req the client request.
     * @param res the response to client.
     * @return void.
     * @exception ServletException In case of a servlet error encountered
     * @exception IOException In case of a I/O error encountered
     */
    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws IOException
    {
        // Initialization
        this.req = req;
        this.res = res;
    }
}

```

```

        // get the session object from the previous servlet
        HttpSession session = req.getSession(true);

        // get now the UserBean object from the session
        freshuser = (UserBean) session.getAttribute("user");

        // Get the user selection from the request
        String FileMenu = req.getParameter("FileMenu");

        // Sets the content type to html text
        res.setContentType("text/html");

        // Gets the PrintWriter object for sending HTML commands
        out = res.getWriter();

        //Generates the title
        out.println("<html><head><title>");
        out.println("Import File Selection" );
        out.println("</title></head>");

        //Generates the body
        out.println("<body bgcolor=\"#fafca3\">");

        // Prints the personal and the security data
        printData();

        // this form gets the url
        out.println( "<form action='FindUrl' method='POST' > " );

        // creates the text box and the submit button
        inputUrl();

        out.println( "</form>" );

        // Create and print the links to other servlets
        exitLinks();

        out.println("</body></html>");
    } // doGet ends here

    /**
     * Displays the available links to direct control
     * in another servlet, in case the user
     * wants to cancel this procedure
     * @return void.
     */
    private void exitLinks()
    {
        out.println("<a href=\"" + res.encodeURL("/metadata/html/login.htm")
            + "\">" + "login again as another user " + "</a>" + " or " + "<br>\n");
    }

```

```

        out.println("<a href=\"" + res.encodeURL("UserOptions")
                    + "\">" + " see again your options" + "</a>" + "<br>\n");
    }

    /**
     * Creates the necessary radio buttons for the
     * server's selection, the text field to get the
     * file and the buttons
     * to clear the field or to submit the filename
     * @return void.
     */
    private void inputUrl()
    {
        String fileUrl="NOT_SELECTED" ;

        out.println(" Choose the server to find the file <br>\n" );
        out.println("   <input type='radio' name='server' value='NPS' checked > NPS   ");
        out.println("   <input type='radio' name='server' value='LOCAL' > LOCAL   ");
        out.println("   <input type='radio' name='server' value='OTHER' > OTHER   ");
        out.println(" and then give the filename or just a word " +
                    " contained in the filename <br>\n" );

        out.println(" <p>       <input type='text' size='40'       name='fileUrl' >");
        out.println("   <input type='submit'   value='Submit' name='B1'>       ");
        out.println("   <input type='reset'     value='Reset'  name='B2'></p>       ");

    }

    /**
     * Prints the user's personal and Security data
     * retrieved from the database that contained
     * in the UserBean object
     * @return void.
     */
    private void printData()
    {
        out.println("<h3>" + " Personal Data " + "</h3>");
        out.println("Last Name : " + "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<strong>" +
freshuser.getLastName() + "</strong><br>\n" +
                    "First Name : " + "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<strong>" +
freshuser.getFirstName() + "</strong><br>\n" );

        out.println("<strong>" + " Security Attributes (presented here only for demo)"
+ "</strong><br>\n");
        out.println(" Service Name   : " + "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<strong>"
+ freshuser.getServiceName() + "</strong><br>\n"
+ " Service Position: " + "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<strong>"
+ freshuser.getServicePos() + "</strong><br>\n"
+ " Service Classification: " + "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<strong>" +

```

```

+         fresheruser.getServiceClass() + "</strong><br>\n"
+
+         " Country ID:" + "&nbsp;&nbsp;&nbsp;<strong>"
+
+         fresheruser.getCountryId() + "</strong><br>\n"
+     );
+ }
+
+ } // ImportFile class ends here

```

```

/*

*****

* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
* Date               :      December 2002

*****

*/

package dbsection;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.awt.*;

/**
 * The ImportToSave class defines a servlet that is responsible to
 * create a web page to help user input the file to be saved,
 * as well as some information related to the file
 */
public class ImportToSave extends HttpServlet implements Serializable
{

    // ***** Data Members
    private UserBean          sessionuser ;
    private Vector            avaFiles ;
    private PrintWriter       out ;
    private HttpServletRequest req;
    private HttpServletResponse res;

    private String[]          classifi = { "TS", "SE", "CL", "UN" } ;
    private String[]          countries = { "ALL", "NOFORN", "NATO", "GRE",
"FRA", "RUS" } ;

    /**
     * Processing user's GET request by simply passing
     * the control to the doPost.
     * @param request The client's request.
     * @param response The response to the client.
     * @return void.
     * @exception ServletException In case of a servlet error encountered
     * @exception IOException In case of a I/O error encountered
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {

```

```

// Simply pass control to doPost()
this.doPost(request, response);
return;
} // end of doGet

/**
 * Processing user's POST request.
 * @param req the client request.
 * @param res the response to client.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a I/O error encountered
 */
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
{
    // Initialization
    this.req = req;
    this.res = res;

    // get the session object from the previous servlet
    HttpSession session = req.getSession(true);

    // get now the UserBean object from the session
    sessionuser = (UserBean) session.getAttribute("user");

    // Sets the content type to html text
    res.setContentType("text/html");

    // Gets the PrintWriter object for sending HTML commands
    out = res.getWriter();

    //Generates the title
    out.println("<html><head><title>");
    out.println("Create new XML file ");
    out.println("</title></head>");

    //Generates the body
    out.println("<body bgcolor=\"#fafca3\">");

    out.println("<FORM ACTION='SaveFile' method='POST'
ENCTYPE='multipart/form-data' > ");

    // Print the Personal and the security data
    printStandardMetadata();

    // input/browse the file from the user
    inputFile();

    // Create and print the links to other servlets

```

```

        exitLinks();

        out.println("</FORM>");
        out.println("</body></html>");

    } // doGet ends here

    /**
     * Displays the available links to direct control
     * in another servlet, in case the user
     * wants to cancel this procedure
     * @return void.
     */
    private void exitLinks()
    {
        out.println("<a href=\"" + res.encodeURL("/metadata/html/login.htm")
                    + "\">" + "login again as another user " + "</a>" + " or " +
"<br>\n");
        out.println("<a href=\"" + res.encodeURL("UserOptions")
                    + "\">" + " see again your options" + "</a>" + "<br>\n");

    }

    /**
     * Creates the necessary text field to help
     * the user to input the file and the buttons
     * to clear the field or to submit the filename
     * @return void.
     */
    private void inputFile()
    {
        out.println("Enter the file below:<BR> ");
        out.println("<INPUT TYPE='file' NAME='fileName' >");
        out.println("<INPUT TYPE='submit' VALUE='Upload file to the server' >");
        out.println("<INPUT TYPE='reset' VALUE='Clear field' >" + "<br>\n");

    }

    /**
     * Prints the user's personal and security data
     * that will appended to the file as its metadata.
     *
     * @return void.
     */
    private void printStandardMetadata()
    {
        Date newDate = new java.util.Date() ;
        // create the table with the standard data
        out.println(" <div align='center'> <center> <table border='2' width='90%'> ");
        // fill in the rows with the known data
        tableRow("CLASSIFIED BY", " ", "DATE ", newDate.toString() );
    }

```

```

        tableRow("Last Name ", sessionuser.getLastName() ,
                "First Name ",
                sessionuser.getFirstName() ) ;
        tableRow("User Service ", sessionuser.getServiceName() ,
                "Position in Service ",sessionuser.getServicePos())
;
        tableRow("User Classification ",sessionuser.getServiceClass() ,
                "User Country ID ", sessionuser.getCountryId() );

        tableRow("File Classification ", dropDownClass(
sessionuser.getServiceClass() ) ,
                "Release to Countries", dropDownCountry(
sessionuser.getCountryId() ) );

        tableRow( "For Official Use Only " ,
"<INPUT TYPE='radio' NAME='FOUO' value='yes' CHECKED >yes"
+
        "<INPUT TYPE='radio' NAME='FOUO' value='no'>no" ,
                "For Internal Use Only " ,
        "<INPUT TYPE='radio' NAME='FIUO' value='yes' CHECKED >yes" +
        "<INPUT TYPE='radio' NAME='FIUO' value='no'>no" );

        out.println(" </table> </center> </div>" );

        out.println("<p align='center'>" + "Before you submit your XML file
you"
                + " have to complete ALL the METADATA fields
below" + " </p>" );

    } // end of printStandardMetadata *****

/**
 * This method creates the allowed options in the drop down list
 * for the classification selection of the file
 * @param userClass The classification of the user,
 * which will be the maximum allowed for the file
 * @return The allowed classifications for the file
 */
private String dropDownClass(String userClass)
{
    // convert user's classification to a number
    int level = 3 ;
    if( userClass.equals("TS") ) level = 0 ;
    if( userClass.equals("SE") ) level = 1 ;
    if( userClass.equals("CL") ) level = 2 ;

    String dropDown = " <SELECT NAME='Class'> " ;

    for (int i=level; i<4 ;i++ )
    {
dropDown = dropDown + "<OPTION VALUE=" + classifi[i] + ">" + classifi[i] ;

```

```

    }

    dropDown = dropDown + " </SELECT> " ;
    return dropDown ;
}

/**
 * This method creates the allowed options in the drop down list
 * for the releasable countries of the file
 * @param userClass The country of the user,
 * @return The allowed countries for the file
 */
private String dropDownCountry(String userCountry)
{
    String dropDown = " <SELECT NAME='Countries'> " ;

    for (int i=0; i<countries.length ;i++ )
    {
        dropDown = dropDown + "<OPTION VALUE=" + countries[i] + ">" +
countries[i] ;

    }
    dropDown = dropDown + " </SELECT> " ;

    return dropDown ;
}

/**
 * This method displays the passed parameters
 * in a table row with specific dimensions
 * @param firstColumn The first column of the row
 * @param secondColumn The second column of the row
 * @param thirdColumn The third column of the row
 * @param fourthColumn The fourth column of the row
 * @return void
 */
private void tableRow(String firstColumn , String secondColumn ,
String thirdColumn , String forthColumn )
{
    out.println("<tr>          <td width='15%'> " + firstColumn + " </td> "
);
    out.println("          <td width='35%'> " + secondColumn + " </td>  " );
    out.println("          <td width='15%'> " + thirdColumn + " </td>  " );
    out.println("          <td width='35%'> " + forthColumn + " </td>  </tr> "
);

}

}

} // ImportToSave class ends here

```

```

/*

*****

* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
* Date               :      December 2002

*****

*/

package dbsection;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import dbsection.*;

/**
 * The class Login defines a servlet that is responsible to authenticate
 * users trying to login. Upon positive authentication,
 * a new user session is created with two attached beans: the customer
 * bean and the database connection module.
 * @author Panos
 */
public class Login extends HttpServlet
{
    // Data members
    private final int DRIVER_ERROR = 1;
    private final int CONNECTION_ERROR = 2;
    private final int QUERY_ERROR = 3;
    private final int QUERY_ERROR_EXC = 5;
    private final int INVALID_USER = 4;

    private HttpServletRequest request;
    private HttpServletResponse response;
    private Connection connection;
    private PrintWriter out;
    private ResultSet rs;
    private ResultSet rs2;

    private DBConnectionBean dbBean;
    private String curServiceName = "non";
    private String curServicePos = "non";
    private String curServiceClass = "non";
    private String curUserCountryId="non";
    private int curUserId = 0;

```

```

        private Exception dbe;
        private UserBean user ;
        private Vector avaFiles ;

/**
 * One-time initialization of the servlet. If the Connection Module is not
 * yet initiated, it ensures its creation.
 * @param config local server configuration parameters.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 */
public void init(ServletConfig config) throws ServletException
{
    super.init(config);

    rs = null;                // for the result set

    DBConnectionBean dbBean =
        (DBConnectionBean)getContext().getAttribute("connectionBean");
    if(dbBean == null)
    {

        dbBean = new DBConnectionBean();    // for the connection module
        dbBean.setDriver("sun.jdbc.odbc.JdbcOdbcDriver");
        dbBean.setUrl("jdbc:odbc:UserDB" );
        dbBean.setUser("panos");
        dbBean.setPassword("greek");

        // Load the driver
        if(!dbBean.isLoaded())
        {
            exitPoint(DRIVER_ERROR);
        }

    }

    this.dbBean = dbBean;

    getContext().setAttribute("connectionBean", dbBean);

} // end of init()

/**
 * Processing user's POST request by simply passing
 * the control to the doGet.
 * @param request The client's request.
 * @param response The response to the client.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a I/O error encountered
 */

```

```

public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
{
    // Simply pass pass control to doGet()
    this.doGet(request, response);
    return;
} // end of doPost

/**
 * Processing user's GET request.
 * @param request the client request.
 * @param response the response to client.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a I/O error encountered
 */
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
{
    // Initialization
    this.request = request;
    this.response = response;

    response.setContentType("text/html");

    out = response.getWriter();

    // Get the login parameters from the request
    String name = request.getParameter("userID");
    String password = request.getParameter("password");

    validateUser(name, password);
} // end of doGet()

/**
 * Authenticates a user with the given username & password
 * @param name The user's username.
 * @param password The user's password.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a servlet error encountered
 */

private void validateUser(String name, String password)
{
    // query the database
    rs = dbBean.query(
        "SELECT * " +
        "FROM User " +
        "WHERE LoginName = '" + name + "'" +

```

```

"AND Password = '" + pass word + "'");

try
{
    if(rs.next())
    {
        // Successful user login
        findNamePos(name, password);

        // initializes the Vector for the files
        avaFiles = new Vector();

        findFiles(curServiceName, curServicePos, curServiceClass,
curUserCountryId);

        // start the session
        startSession();
    }
    else
    {
        // Invalid login data
        exitPoint(INVALID_USER);
    }

    rs.close(); // close the ResultSet
}
catch(Exception e)
{
    exitPoint(QUERY_ERROR_EXC);
}

} // end of validateUser()

/**
 * Finds the Security authorizations of a valid user
 * @param name          The user's username.
 * @param password      The user's password.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a servlet error encountered
 */
private void findNamePos(String name, String password)
{
    ResultSet rs3=dbBean.query(
        "SELECT * " +
        "FROM User " +
        "WHERE LoginName = '" + name + "'" +

```

```

        "AND Password = '" + password + "'");
    try
    {
        if(rs3.next())
        {
            curServiceName      = rs3.getString("ServiceName");
            curServicePos       = rs3.getString("ServicePos");
            curServiceClass     = rs3.getString("ServiceClass");
            curUserCountryId    = rs3.getString("CountryId");
            curUserId           = Integer.parseInt(
rs3.getString("UserID"));
        }
    }
    catch(Exception e)
    {
        exitPoint(QUERY_ERROR_EXC);
        dbe = e;
    }
}

/**
 * Finds the files that are available to a specific user according
 * to his/her security authorizations
 * @param service      The user's service
 * @param position     The user's position in his or her service
 * @param clearance    The user's clearance (Unclassified, Classified etc.)
 * @param country      The user's country
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of an I/O error encountered
 */
private void findFiles( String service, String position,
                        String clearance, String country)
{
    // retrieve the available files according to the below criteria
    ResultSet rs4=dbBean.query(
        "SELECT * " +
        "FROM File " +
        "WHERE UserServiceName = '" + service + "' +
        "AND UserServicePos  = '" + position + "' +
        "AND UserServiceClass = '" + clearance + "' +
        "AND UserCountryId  = '" + country + "'");
    try
    {
        while(rs4.next())
        {
            avaFiles.add( rs4.getString("File") );
        }
    }
    catch(Exception e)
    {

```

```

        exitPoint(QUERY_ERROR_EXC);
    }

    } // findFiles

/**
 * Starts a new session after a user
 *      is succefully authenticated.
 * @return void
 * @exception Exception In case of an invalid situation is encountered
 */
    private void startSession() throws Exception
    {

        // The next servlet to be called through a RequestDispatcher object
        // Get the dispatcher
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher("/UserOptions");

        if (dispatcher == null)
        {
            // No dispatcher means the given file could not be found
            response.sendError(response.SC_NO_CONTENT);
        }
        else
        {
            // Get or start a new session for this user
            HttpSession session = request.getSession(true);
            if(!session.isNew())
            {
                // Ensure the session is newly created
                session.invalidate();
                session = request.getSession(true);
            }

            // get all the data from the database and create a new User object
            user = createUser() ;

            // Close the database connection
            dbBean.isClose();

            // Attach customer bean to this session object
            session.setAttribute("user", user);

            // Pass control to a different page
            dispatcher.forward(request, response);
        }

    } // end of startSession()

/**

```

```

* Creates a new userBean object from the
*     information retrieved from the database.
* @return A UserBean object containing all the user's data is returned.
* @exception Exception In the case of an invalid data
*/

private UserBean createUser() throws Exception
{
    UserBean user = new UserBean();

    user.setUserID(rs.getInt("UserID"));
    user.setFirstName(rs.getString("Fname"));
    user.setMiddleInitial(rs.getString("Mi"));
    user.setLastName(rs.getString("Lname"));
    user.setStreet(rs.getString("Street"));
    user.setCity(rs.getString("City"));
    user.setZip(rs.getString("Zip"));
    user.setState(rs.getString("State"));
    user.setPhone(rs.getString("Phone"));
    user.setLoginName(rs.getString("LoginName"));
    user.setPassword(rs.getString("Password"));
    user.setEmail(rs.getString("Email"));

    user.setServiceName(rs.getString("ServiceName"));
    user.setServicePos(rs.getString("ServicePos"));
    user.setServiceClass(rs.getString("ServiceClass"));
    user.setCountryId(rs.getString("CountryId"));

    user.setAvaFiles(avaFiles);

    return user;
}

/**
* Defines several types of exit conditions
*     depending on the specified error
*
* @param exitCondition That integer specifies the error causes the exit.
* @return void.
*/
private void exitPoint(int exitCondition)
{
    String output = new String();

    switch(exitCondition)
    {
        case CONNECTION_ERROR:
            output += "Application error: unable to establish connection to database";
            break;

        case QUERY_ERROR:

```

```

        output += "Application error: invalid database query";
        break;

    case INVALID_USER:
        output += "Invalid username and/or password";
        break;

    default:
        output += "Application error";

    } // end switch

    out.println("<body bgcolor=\"\#fafca3\">");
    out.println("<H3>Metadata Security Label Tags Authentication Process</H3>");

    out.println("<FONT COLOR='FF0000'>" + "*** ERROR ENCOUNTERED
*** " + "<br>\n");
    out.println( output + "<br>\n");
    out.println("Please choose from the following" + "</FONT>" +
"<br>\n");

        out.println("<a href=\"" +
response.encodeURL("/metadata/html/login.htm")
+ "\">" + "Try to login again " + "</a>" +
"<br>\n");

        out.println("<a href=\"" +
response.encodeURL("/metadata/html/main.htm")
+ "\">" + "Home page" + "</a>" + "<br>\n");

    out.println("<P>\n");
    out.println("</BODY>\n");
    out.println("</HTML>");

    } // end of exitPoint()

} // end of Login class

```

```

/*

*****

* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
*   Date             :      December 2002

*****

*/

package dbsection;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;

/**
 * This class defines a servlet that logout
 * a user and end his/her session.
 * @author Panos
 */
public class Logout extends HttpServlet
{
    // Data members
    private final int ERROR_SESSION = 1;
    private HttpServletRequest request;
    private HttpServletResponse response;
    private PrintWriter out;
    private ResultSet rs;
    private DBConnectionBean dbBean;
    private CustomerBean customer;
    private String output = new String();

    /**
     * Processing user's POST request by simply passing
     * the control to the doGet.
     * @param request The client's request.
     * @param response The response to the client.
     * @return void.
     * @exception ServletException In case of a servlet error encountered
     * @exception IOException In case of a I/O error encountered
     */
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        // Simply pass pass control to doGet()
        this.doGet(request, response);
        return;
    }
}

```

```

    } // end of doPost

    /**
     * Processing user's GET request.
     * @param request the client request.
     * @param response the response to client.
     * @return void.
     * @exception ServletException In case of a servlet error encountered
     * @exception IOException In case of a I/O error encountered
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        // Startup settings
        this.request = request;
        this.response = response;

        // Sets the content type to html text
        response.setContentType("text /html");

        // Gets the PrintWriter object for sending HTML commands
        out = response.getWriter();

        //Generates the body
        out.println("<body bgcolor=\"#fafca3\">");

        output = "";
        processRequest();

    } // end of doGet()

    /**
     * Processing the request for logging the user out
     * and terminating the current user session.
     * @return void
     */
    private void processRequest()
    {
        // Terminate the session for this user
        HttpSession session = request.getSession(false);
        if(session != null)
        {
            session.invalidate();
        }

        out.println("Thank you for using Metadata Security Label Tags System."
+ "<br>\n");

        exitLinks();
    }

```

```

    } // end of processRequest()

    /**
     * Displays the available links to direct control
     * in another page, in case the user
     * wants to discontinue the session
     * @return void.
     */
    private void exitLinks()
    {
        out.println( "<br> <br> <br>\n");
        out.println("<a href=" +
response.encodeURL("/metadata/html/main.htm")
                        + ">" + "Home page" + "</a>" + "<br>\n");

        out.println("<a href=" +
response.encodeURL("/metadata/html/login.htm")
                        + ">" + "Login again as another user " + "</a>"
+ " or " + "<br>\n");
    }

} // end of Logout class

```

```

/*
*****
* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
* Date               :      December 2002
*****

*/
package dbsection;

import java.io.*;
import java.io.Writer ;

/**
 * The MetaTags class codes the tags of
 * a file as an object
 * @author Panos
 */
public class MetaTags
{

    String[] parsedFile ;
    int index ;

    /**
     * The default constructor of the class
     * initializes the variables.
     */
    public MetaTags()
    {
        parsedFile = new String[120] ;
        index=0 ;

    }

    /**
     * Gets the tags of the parsed file
     * @return An array of String representing
     * the tags of the file
     */
    public String[] getParsedFile()
    {
        return parsedFile ;

    }

    /**
     * Adds a new tag passed as the parameter
     * @param newString The new tag to be added

```

```
* @return void
*/
    public void add(String newString)
    {
        index = index +1 ;
        parsedFile[index] = newString ;
    }
}
```

```

/*

*****
* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor       :      Tim Levin
* Date               :      December 2002
*****

*/

package dbsection;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;

/**
 * The class OpenFile defines a servlet that is responsible to
 * open any file a user may select either the file is saved
 * locally or in another server
 * @author Panos
 */

public class OpenFile extends HttpServlet
{

    // Data Members
    private UserBean newuser ;

    private Vector avaUNFiles ;
    private Vector avaCLFiles ;
    private Vector avaSEFiles ;
    private Vector avaTSFiles ;

    private PrintWriter out ;
    private DBConnectionBean dbBean ;

    private String[] classification = { "UN" , "CL" , "SE" , "TS" } ;
    private String[] date = new String [10] ;

    private HttpServletRequest req;
    private HttpServletResponse res;

    private String selfFile ;
    private String classif ;
    private String action ;
    private String otherServer ;

```

```

        private Vector avaFiles ;

/**
 * Processing user's GET request by simply passing
 * the control to the doPost.
 * @param request The client's request.
 * @param response The response to the client.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a I/O error encountered
 */
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
{
    // Simply pass pass control to doPost()
    this.doPost(request, response);
    return;
} // end of doGet

/**
 * Processing user's POST request.
 * @param request the client request.
 * @param response the response to client.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a I/O error encountered
 */
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws IOException
{
    // Initialization
    this.req = req;
    this.res = res;

    // get the session object from the previous servlet
    HttpSession session = req.getSession(true);

    // get now the UserBean object from the session
    newuser = (UserBean) session.getAttribute("user");

    // Get the user selection from the request
    String FileMenu = req.getParameter("FileMenu");

    // Sets the content type to html text
    res.setContentType("text/html");

    // Gets the PrintWriter object for sending HTML commands
    out = res.getWriter();

```

```

//Generates the title
out.println("<html><head><title>");
out.println("User Options !!!!!" );
out.println("</title></head>");

//Generates the body
out.println("<body bgcolor=\"#fafca3\">");

    // Print the Personal and the security data
    printData();

    selFile = req.getParameter("selFile");
    classif = req.getParameter("classification");
    action = req.getParameter("action");

    // this is the first time passing the serv let
    // from the local server
    if( selFile.equals("start") )
    {
        // Print the available files for the specific user
        printAvaFiles();
    }

    // this is the first time passing the servlet
    // from another associated server
    else if ( selFile.equals("server" ) )
    {
        // Print the available files sent by the other server
        printServerAvaFiles();
    }

    // this is the second time passing the servlet
    // so give the choices how to open the files on local server
    else if ( action.equals("menu") )
    {
        out.println("<br>\n" + "Selected file : " + selFile + "<br>\n" );
        out.println("If you want to open the file with your own XML Editor " +
"<br>\n");
        out.println("or to save it for future use " );
        out.println("<a href=\"" + res.encodeURL("/metadata/xml/" + classif + "/"
+ selFile )
+ "\">" + " click here " + "</a><br>\n" );

        action = "startEditor" ;

        out.println("<br>\n" + "If you want to open the file with an applet -based XML
Editor "
+ "<br>\n" );

        out.println("<a href=\"" + res.encodeURL("OpenFile?classification="

```

```

selfFile)
        + classif + "&action=" + action + "&selfFile="+
        + ">" + " click here " + "</a><br>\n" );
    }

    // this is the second time passing the servlet
    // so give the choises how to open the files on the associated server

    else if ( action.equals("menuServer") )
    {

        out.println("<br>\n" + "Selected file : " + selfFile + "<br>\n" );

        out.println("If you want to open the file with your own XML Editor " + "<br>\n" );
        out.println("or to save it for future use " );

        // get the last element which contains the name of the parent server
        otherServer = (String) avaFiles.lastElement() ;
        otherServer = otherServer.substring( 0 , otherServer.length()-8 ) ;
        selfFile = selfFile.substring(2,selfFile.length() ) ;

        out.println(" **** server name = " + otherServer + " = clasiff = " + classif );
        out.println(" = and the filename is = " + selfFile );
        out.println("<a href=" + res.encodeURL(otherServer + "xml/" + classif + "/" + selfFile )
            + ">" + " click here " + "</a><br>\n" );

        action = "startEditor" ;

        out.println("<br>\n" + "If you want to open the file with an applet -based XML Editor "
            + "<br>\n" );
        out.println("<a href=" + res.encodeURL("OpenFile?classification="
            + classif + "&action=" + action + "&selfFile="+selfFile)
            + ">" + " click here " + "</a><br>\n" );

    }

    // this is the final time passing the servlet
    // which simply initiates the applet based XML Editor
    else
    {
        startEditor(classif, selfFile) ;
    }

    // print the exit options
    out.println("<a href=" + res.encodeURL("/metadata/html/login.htm")
        + ">" + "login again as another user " + "</a><br>\n" +
    " or ");

    out.println("<a href=" + res.encodeURL("UserOptions")
        + ">" + " see again your options" + "</a>" + "<br>\n");
    out.println("</body></html>");

```

```

} // doGet ends here

/**
 * Calls the applet that will in turn
 * load and initiate the applet-based XML editor
 * @param classLevel The classification level of the user that will help
 * find the directory where the file is saved.
 * @param sFile The name of the file that the editor will open
 * @return void.
 */
private void startEditor(String classLevel, String sFile)
{
    out.println("<applet codebase=/metadata/applets/ code=MainApplet.class
width=10 height=10>");

    out.println("<PARAM NAME=reqUrl VALUE='
http://localhost:8080/metadata/xml/"
+ classLevel + "/" + sFile + "'>");

    out.println("</applet><br>\n");
}

/**
 * Converts user's classification to a number
 * @param classification The classification level of the user
 * @return An integer number representing the user's classification
 */
private int getClassLevel(String classification)
{
    // convert user's classification to a number
    int level = 0;
    if( classification.equals("TS") ) level = 3;
    if( classification.equals("SE") ) level = 2;
    if( classification.equals("CL") ) level = 1;

    return level;
}

/**
 * This method gets the available files passed
 * from the associated server and present them
 * to the user according to his/her classification
 * @return void
 */
private void printServerAvaFiles()
{
    out.println("Click on the links below to open the respective file " + "<br>\n");

    // initialize a Vector to get them
    avaFiles = new Vector();

```

```

// user allowed to see files only up to his/her classification
for (int i=0; i<= getClassLevel( newuser.getServiceClass() ) ; i++)
{
    avaFiles = newuser.getAvaFiles() ;

    String aFile;
    String action = "menuServer" ;

    // get the last element which contains
    // the name of the parent server
    otherServer = (String) avaFiles.lastElement() ;

    out.println( "<hr/>" );
    out.println("Files at " + otherServer + " with classification : "
        + classification[i] + "<br>\n" );

    for(int j=0; j<avaFiles.size() - 1 ; j++ )
    {
        aFile = (String) avaFiles.elementAt(j) ;

        if( getClassLevel( aFile.substring(0,2) ) == i )
        {
            date = findVersion(aFile) ;

            out.println("<a href=" +
res.encodeURL("OpenFile?classification="
+ classification[i] +
"&action=" + action
+ "&selFile="+ aFile) +
">" + date[6] + "</a>" );

            out.println(".....Version Date : " + date[0] + "/" + date[1]
                + "/" + date[2] );
            out.println(".....time : " + date[3] + ":" + date[4] + ":"
                + date[5] + "<br>\n" );
        }
    } // j loop
} // i loop

// insert a horizontal line
out.println( "<hr/>" );

} // printServerAvaFiles ends here

/**
 * This method gets the available files through
 * the findFiles method and display them
 * to the user according to his/her classification
 * @return void

```

```

*/
private void printAvaFiles()
{
    out.println("Click on the links below to open the respective file " + "<br>\n" );

    // initialize a Vector to get them
    Vector avaFiles = new Vector() ;
    // opens a connection to the database
    // to make the query and retrieve the most
    // updated files
    openConnection() ;

    // user allowed to see files only up to his/her classification
    for (int i=0; i<= getClassLevel( newuser.getServiceClass() ) ; i++)
    {

        avaFiles = findFiles ( newuser.getServiceName() ,
newuser.getServicePos() ,
                                                                    classification[i] ,
newuser.getCountryId() ) ;

        String aFile;
        String action = "menu" ;
        Enumeration enum = avaFiles.elements();

        out.println( "<hr/>" );
        out.println("Files with classification : " + classification[i] + "<br>\n" );

        while(enum.hasMoreElements() )
        {
            aFile = (String) enum.nextElement();
            date = findVersion(aFile) ;

            out.println("<a href=" +
res.encodeURL("OpenFile?classification="
                                                                    + classification[i] + "&action="
+ action
                                                                    + "&selFile=" + aFile) + ">" +
date[6] + "</a>" );

            out.println(".....Version Date : " + date[0] + "/" + date[1] + "/" +
date[2] );
            out.println(".....time : " + date[3] + ":" + date[4] + ":" + date[5] +
"<br>\n" );

        } // while loop ends here

    } // i loop ends here

    // insert a horizontal line
    out.println( "<hr/>" );

```

```

    } // printAvaFiles ends here

/**
 * This method analyses the filename to extract
 * the date and the time, the file saved in the system
 * which actually represents the version of the file.
 * @param fullName The complete name of the file
 * @return An array of string containing the date and the time
 */
private String[] findVersion(String fullName )
{
    char letter ;
    int x =0 ;
    String[] date = { "", "", "", "", "", "", "" } ;

    for(int i=0; i<fullName.length() ; i++)
    {
        letter = fullName.charAt(i) ;

        if (letter != '~') date[x] = date[x] + fullName.substring(i, i+1) ;
        else if (letter == '~') x++ ;

    }

    return date ;
}

/**
 * Finds the files that are available
 * to a specific user according to his security
 * authorizations
 * @param service The user's service
 * @param position The user's position in his or her service
 * @param clearance The user's clearance (Unclassified, Classified etc.)
 * @param country The user's country
 * @exception Exception In case of a error encountered
 * @return void.
 */

private Vector findFiles( String service, String position,
String
clearance, String country)
{
    Vector avFiles = new Vector() ;

    // retrieve the available files according to the below criteria
    ResultSet rs=dbBean.query(
        "SELECT * " +
        "FROM File " +
        "WHERE UserServiceName = " + service + " + "" +
        "AND UserServicePos = " + position + " + "" +

```

```

        "AND UserServiceClass = '" + clearance + "'" +
        "AND UserCountryId = '" + country + "'");

        try
        {

                while(rs.next())
                {
                        avFiles.add( rs.getString("File") );
                }

        }
        catch(Exception e)
        {
                out.println("The system could not process your request " );
                out.println(" due to an exception occured. Please try again " + "<br>\n" );

                out.println("<a href='" + res.encodeURL("UserOptions")
                        + "'>" + "Go back to your options" + "</a>" + "<br>\n");

        }

        return avFiles ;

    } // findFiles

/**
 * Opens a connection with the database to retrieve
 * the files that are available to the user
 * @return void
 */
private void openConnection()
{
        dbBean =
(DBConnectionBean)getContext().getAttribute("connectionBean");
        if(dbBean == null)
        {

                dbBean = new DBConnectionBean();    // for the connection module
                dbBean.setDriver("sun.jdbc.odbc.JdbcOdbcDriver");
                dbBean.setUrl("jdbc:odbc:UserDB" );
                dbBean.setUser("panos");
                dbBean.setPassword("greek");

                // Load the driver
                if(!dbBean.isLoaded())
                {
                        out.println("The system could not process your request " );
                        out.println(" due to an error to the Database Driver."
                                + " Please try again " + "<br>\n"

);

```

```

        out.println("<a href=\"" + res.encodeURL("UserOptions")
            + "\">" + "Go back to your options" + "</a>" + "<br>\n");
    }
}

this.dbBean = dbBean;

getServletContext().setAttribute("connectionBean", dbBean);
}

/**
 * Prints the user's personal and Security data
 * retrieved from the database that contained
 * in the UserBean object
 * @return void.
 */
private void printData()
{
    out.println("<h3>" + " Personal Data " + "</h3>");
    out.println("Last Name : " + "&nbsp;&nbsp;&nbsp;<strong>" +
        newuser.getLastName() + "</strong><br>\n" +
        "First Name : " + "&nbsp;&nbsp;&nbsp;<strong>" +
        newuser.getFirstName() + "</strong><br>\n");

    out.println("<strong>" + " Security Attributes (presented here only for demo)" +
        "</strong><br>\n");
    out.println(" Service Name : " + "&nbsp;&nbsp;&nbsp;<strong>" +
        newuser.getServiceName() + "</strong><br>\n" +
        " Service Position:" + "&nbsp;&nbsp;&nbsp;<strong>" +
        newuser.getServicePos() + "</strong><br>\n" +
        " Service Classification:" + "&nbsp;&nbsp;&nbsp;<strong>" +
        newuser.getServiceClass() + "</strong><br>\n" +
        " Country ID:" + "&nbsp;&nbsp;&nbsp;<strong>" +
        newuser.getCountryId() + "</strong><br>\n");
}

} // OpenFile class ends here

```

```
/*
```

```
*****  
* Master Thesis      :      Metadata Security Label Tags  
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)  
*   Advisor          :      Ted Lewis, Ph.D.  
*   2nd Advisor      :      Tim Levin  
* Date               :      December 2002
```

```
*****
```

```
*/
```

```
package dbsection;
```

```
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;  
import java.net.*;  
import java.util.*;  
import javax.activation.*;  
import dbsection.*;
```

```
/**
```

```
 *      The SaveFile class is responsible to upload the file  
 * from the user's machine to a temporary area of the system  
 * and to check and parse the file  
 *
```

```
*/
```

```
public class SaveFile extends HttpServlet implements Serializable  
{
```

```
    // Data Members
```

```
    private UserBean      user;  
    private HttpServletRequest req;  
    private HttpServletResponse res;
```

```
    private String        uploaded;  
    private File           selectedFile ;  
    private File           file ;  
    private PrintWriter out ;
```

```
    private int            MAXIMUM_FILE_LINES = 300 ;  
    private boolean        decision=false ;  
    private String         desiredClass = null ;  
    private String         desiredCountry = null ;  
    private String         fouo = null ;  
    private String         fiuo = null ;
```

```
    private String         tempFile ;  
    private String         filename ;  
    private String         ending ;  
    private String         errorState ;
```

```

        private String []      fileSecAttr ;

        static      FileTags      fileTags ;
        static      SecMetadataManager  manager ;

/**
 * Processing user's POST request.
 * @param request The client request.
 * @param response The response to client.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a I/O error encountered
 */
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws IOException
{
    // Initialization
    this.req = req;
    this.res = res;

    // get the session object from the previous servlet
    HttpSession session = req.getSession(true);

    // get now the UserBean object from the session
    user = (UserBean) session.getAttribute("user");

    // Sets the content type to html text
    res.setContentType("text/html");

    // Gets the PrintWriter object for sending HTML commands
    out = res.getWriter();

    //Generates the title
    out.println("<html><head><title>");
    out.println("Get the User's File to Save it --- servlet" );
    out.println("</title></head>");

    //Generates the body
    out.println("<body bgcolor=\"#fafca3\">");

        // Print the Personal and the security data
        printData();
        errorState = "" ;

        try
        {
            out.println(" ..... " + "<br>\n");
            out.println(" Uploading file to the server side ..... " + "<br>\n");

                uploaded = uploadFile() ;

        }

```

```

        catch (IOException e)
        {
            // go back in case the file is NOT an XML
            errorGoBack("COULD NOT BE UPLOADED DUE TO A COMMUNICATION
PROBLEM ");
        }

        // If everything was fine
        if( errorState.equals("") )
        {
            out.println(" .....File was successfully uploaded" + "<br>\n");
            out.println(" ..... " + "<br>\n");
            out.println(" Parsing the file ..... " + "<br>\n");
            checkingFile(uploaded) ;
        }

        // Create and print the links to other servlets
        exitLinks();

        out.println("</body></html>");
    } // doPost ends here

    /**
     * Uploads the file to a temporary position
     * for further processing
     * @return If everything was done successfully
     *         returns the name of the temporary file
     * @exception IOException In case a communication exception is thrown
     */
    private String uploadFile() throws IOException
    {
        // this Reader gets the file from the request
        BufferedReader bufReader = req.getReader();
        String str = null ;

        // those three lines are used to specify the correct
        // user directory because it is different in every server
        String userDir=System.getProperty("user.dir") ;
        int len = userDir.length() ;
        if( userDir.substring(len -3 ,len).equals("bin") )
            System.setProperty("user.dir" , "C:\\jakarta-tomcat-4.1" ) ;

        // save the file in that temporary address in the server side
        tempFile = System.getProperty("user.dir")
            + File.separatorChar + "webapps"
            + File.separatorChar + "metadata"
            + File.separatorChar + "xml"
            + File.separatorChar + "temporary"
            + File.separatorChar + user.getUserID()

        +"temp -1.xml" ;
    }

```

```

File outFile = new File(tempFile);
FileOutputStream outFileStream = new
FileOutputStream(outFile);
PrintWriter outputStream = new
PrintWriter(outFileStream);

int firstLine = 0;
String xmlStartStr = "";
String xmlEndStr = "";

// get the parameters from the beginning of the stream
for(int j=0; j<4; j++) desiredClass = bufReader.readLine(); //
desired classification
for(int j=0; j<4; j++) desiredCountry = bufReader.readLine();
// releasable to countries
for(int j=0; j<4; j++) fouo = bufReader.readLine();
// For Official Use Only
for(int j=0; j<4; j++) fiuo = bufReader.readLine();
// For Internal Use Only

// string containing the session number
str = bufReader.readLine();

// string containing the filename
str = bufReader.readLine();

// extract the name of the file
filename = findFilename(str);

if(!errorState.equals(""))
{
    // go back in case the file has NOT an XML file extension
    // or an empty filename
    errorGoBack(errorState);
}
else
{

    // find the beginning of the document in the first 20 lines
    for (int i=0; i<20; i++)
    {
        str = bufReader.readLine();

        if( !str.equals("") && str.length()>5 ) xmlStartStr =

str.substring(0,5);

        if( xmlStartStr.equals("<?xml") )
        {
            // the first line is found
            firstLine=i;

```

```

        i=20 ; // exit the loop
        outputStream.println(str) ; // write the first line
    } // if ending

    } // i loop

    // write the file to the temporary address
    for (int i=firstLine; i<MAXIMUM_FILE_LINES ; i++)
    {
        str = bufReader.readLine() ;

        // trying to detect the end of the file
        if( !str.equals("") && str.length()>10 ) xmlEndStr =
str.substring(0,10) ;

        if(!xmlEndStr.equals("-----") )
        {
            outputStream.println(str) ;
        }
        else
        {
            i=MAXIMUM_FILE_LINES ;
        }
    } // i loop

    } // else ends here

    outputStream.close();

    return tempFile ;

    } // uploadFile ends here

/**
 * This method extracts the name of the file from
 * incoming stream, and checks if it is a valid XML filename
 * @param stringFile The complete name of the file
 * @return The name of the file found
 */
private String findFilename(String stringFile)
{
    String filename ;

    int len = stringFile.length() ;
    int pos =0 ;
    int flag = 0 ; // this flag is used in some browsers where they
get // only the file name and not the
whole path + filename

    ending = stringFile.substring( (len-4) , (len-1) ) ;

```

```

        boolean emptyFilename = true;

        // find the filename
        for (int i=(len-1); i>0; i--)
        {
            if( stringFile.charAt(i) == "" ) flag = flag + 1 ;

            if( stringFile.charAt(i) == File.separatorChar || flag == 2)
            {
                pos = i+1 ;
                i=0 ;
                emptyFilename = false ;
            }
        }

        filename = stringFile.substring( pos , (len-1) ) ;

        if(!ending.equals("xml")) errorState="IS NOT AN XML FILE" ;

        if(emptyFilename==true)
        {
            errorState="HAS NO NAME !!!!!" ;
            filename="" ;
        }

        return filename ;

    }

    /**
     * Defines several types of error conditions
     * depending on the specified error
     * @param error The error specifies the cause of the problem.
     * @return void.
     */
    private void errorGoBack(String error)
    {
        out.println("<FONT COLOR='#FF0000'>" + "**** ERROR ENCOUNTERED **** " +
"<br>\n");
        out.println("THE FILE YOU ENTERED --> " + filename + "<br>\n");
        out.println( error + "<br>\n");
        out.println("Please " + "</FONT>" + "<a href="
            + res.encodeURL("CreateFile") + ">"
            + " go back and select another file" + "</a>" + "<br>\n");
    }

    /**
     * This method passes the file to the Echo24 class
     * which parses the file and returns a filetags object
     * This object and the userBean object are passed to

```

```

* the security manager for the decision.
* @param selFile the selected file to be parsed
* @return void.
*/
private void checkingFile(String selFile)
{
    // initialize a new Echo24 object
    Echo24 echo24 = new Echo24() ;

    // create the FileTags object for that specific filename
    fileTags = echo24.parseFile(selFile) ;

    // if the file is not a valid XML file
    // then a null fileTags object is returned
    if(fileTags==null)
    {
        errorGoBack("IS NOT A VALID XML FILE. VALIDATE IT
WITH THE IC SECURITY MARKINGS");
    }

    // the file is parsed successfully
    else
    {
        // read file's sec attr
        fileSecAttr = fileTags.getSecAttributeValues() ;

        out.println(" .....File was parsed successfully !!!! " + "<br>\n");
        out.println(" ..... " + "<br>\n");
        out.println(" Comparing File's Security Metadata Labels " + "<br>\n");

        // initialize a new security metadata manager
        // for that specific file and user
        manager = new SecMetadataManager( fileTags , user) ;

        // make the comparison through the compareTags method
        String status = manager.compareTags(desiredClass,
desiredCountry, fouo , fiuo) ;

        // "normal" case.
        if( status.equals("normal") )
        {
            out.println(" The whole process was successful" );
            out.println("<a href=" +

res.encodeURL("UpdateDb?desiredClass=" + desiredClass +
"&filename=" + filename ) + ">" +
"Confirm " + "</a>");
            out.println("to update the database and save the file to
the server " );
        }
    }
}

```

```

        // "upgrade" or "degrade" case.
        else if( status.equals("upgrade") || status.equals("downgrade") )
        {
            // ask for confirmation
            askConfirm(status);
        }

        // there is a problem with the labels and
        // the process could not be accomplished
        else
        {
            out.println(" **** WARNING *** There is a problem
with the labels !!!!" );
            out.println("
***** " );
            out.println(" Please validate your file with the IC
security metadata" + "<a href=\"" +
res.encodeURL("CreateFile") +
">" + " and try again " + "</a>" + "<br>\n");
        }

    } // else fileTags is NOT null

} // checkingFile method

/**
 * In the cases where the status found is "upgrade" or "downgrade"
 * this method displays the appropriate message
 * @param status The results of the tag's comparison
 * @return void.
 */
private void askConfirm(String status)
{
    out.println(" ..... " + "<br>\n");
    out.println(" **** WARNING *** The file will be " + status +
"d *****" + "<br>\n" );
    out.println(" ***** The file is already marked as " +
fileSecAttr[0] + "<br>\n");
    out.println(" ***** and you have asked to be saved as " +
desiredClass + "<br>\n");
    out.println(" ..... " + "<br>\n");
    out.println(" If you agree " + "<a href=\"" +
res.encodeURL("UpdateDb?desiredClass=" + desiredClass +
"&filename=" + filename ) + ">" +
" confirm " + "</a>" + "<br>\n");

    out.println(" If you want to make a change " + "<a href=\"" +
res.encodeURL("ImportToSave") + ">"
+ " go back " + "</a>" + "<br>\n");

```

```

    }

    /**
     * Displays the available links to direct control
     * in another servlet, in case the user
     * wants to cancel this procedure
     * @return void.
     */
    private void exitLinks()
    {
        out.println( "<br> <br> <br>\n");
        out.println("<a href=" + res.encodeURL("/metadata/html/login.htm")
                    + ">" + "login again as another user " + "</a>" +
" or " + "<br>\n");
        out.println("<a href=" + res.encodeURL("UserOptions")
                    + ">" + " see again your options" + "</a>" +
"<br>\n");
    }

    /**
     * Prints the user's personal and Security data
     * retrieved from the database that contained
     * in the UserBean object
     * @return void.
     */
    private void printData()
    {
        out.println("<h3>" + " Personal Data " + "</h3>");
        out.println("Last Name : " + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getLastName() + "</strong><br>\n" +
"First Name : " + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getFirstName() + "</strong><br>\n" );

        out.println("<strong>" + " Security Attributes (presented here only for
demo)" + "</strong><br>\n" );

        out.println(" Service Name : " + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getServiceName() + "</strong><br>\n" +
" Service Position:" + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getServicePos() + "</strong><br>\n" +
" Service Classification:" + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getServiceClass() + "</strong><br>\n" +
" Country ID:" + "&nbsp;&nbsp;&nbsp;<strong>" +
user.getCountryId() + "</strong><br>\n"
);
    }

}

} // SaveFile class ends here

```

```

/*
*****
* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
* Date               :      December 2002
*****

*/

package dbsection;

import dbsection.FileTags ;

/**
 * The class SecMetadataManager is responsible to compare the
 * received UserBean and FileTags objects and to produce
 * the necessary decision according to the policy
 * @author Panos
 */

public class SecMetadataManager
{
    // Data Members
    private FileTags      fileTags ;
    private UserBean      userBean ;
    private boolean approved ;

    /**
     * The default constructor of the class
     * receives the passed objects and initializes
     * the variable of the decision as false
     * @param tags    The FileTags object representing the
     *                tags of a specific file
     * @param UserBean The UserBean object which will be compared
     */
    public SecMetadataManager(FileTags tags , UserBean user )
    {
        fileTags = tags ;
        userBean = user ;
        approved = false ;
    }

    /**
     * This method makes the decision
     * @return A boolean variable containing the decision
     */
    public boolean makeDecision()

```

```

        {
            String[] secAttribute =fileTags.getSecAttributes() ;
            String[] secAttributeValue =fileTags.getSecAttributeValues() ;

            if( secAttributeValue[0].equals( userBean.getServiceClass() ) ) approved=true ;
            return approved ;

        } // makeDecision ends here

/**
 * This method compares the tags of the file with those
 * of the user, and returns the result.
 * @return A String that can be upgrade, downgrade or normal
 */
    public String compareTags(String desiredClass , String desiredCountry ,
                               String fouo , String fiuo )
    {
        String status = null ;
        String[] secAttribute =fileTags.getSecAttributes() ;
        String[] secAttributeValue =fileTags.getSecAttributeValues() ;

        int fileLevel = getClassLevel( secAttributeValue[0] ) ;
        int userLevel = getClassLevel( userBean.getServiceClass() ) ;
        int desiredLevel = getClassLevel( desiredClass ) ;

        if(fileLevel==desiredLevel && userLevel<=fileLevel) status ="normal" ;
        if(fileLevel>desiredLevel) status ="upgrade" ;
        if(fileLevel<desiredLevel) status ="downgrade" ;

        return status ;
    } // compareTags ends here

/**
 * This method converts a classification level to a number
 * to be easiest to compare
 * @return An integer representing the classification level
 */
    private int getClassLevel(String classification)
    {
        // convert user's classification to a number
        int level = 3 ;
        if( classification.equals("TS") ) level = 0 ;
        if( classification.equals("S") ) level = 1 ;
        if( classification.equals("C") ) level = 2 ;

        return level ;
    }

}

```

```

/*

*****
* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
* Date               :      December 2002
*****

*/

package dbsection;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.util.Vector;
import java.util.Enumeraion ;
import java.sql.*;
import java.lang.* ;
import dbsection.*;

/**
 * The class ServerLogin defines a servlet that is responsible to authenticate
 * associated servers trying to login. Upon positive authentication,
 * all the names of the files that are available to that specific
 * server are sent for further processing.
 * @author Panos
 */

public class ServerLogin extends HttpServlet
{
    // Data members
    private final int DRIVER_ERROR = 1;
    private final int CONNECTION_ERROR = 2;
    private final int QUERY_ERROR = 3;
    private final int QUERY_ERROR_EXC = 5;
    private final int INVALID_USER = 4;

    private HttpServletRequest request;
    private HttpServletResponse response;
    private Connection connection;
    private PrintWriter out;

    private ServletOutputStream serverOut ;
    private ResultSet rs;
    private ResultSet rs2;
    private DBConnectionBean dbBean;

```

```

        private String curServiceName  ="non";
        private String curServicePos   ="non" ;
        private String curServiceClass ="non";
        private String curUserCountryId="non";
        private int   curUserId        =0 ;

        public Exception dbe;
        public UserBean user ;
        public Vector avaFiles ;
        private Enumeration enum ;

        private String serverName ;
        private String serverPass ;
        private String filename ;
        private String server ;
        private String[] savedFilename = new String [10] ;

/**
 * Processing user's POST request by simply pass ing
 * the control to the doGet.
 * @param request The client's request.
 * @param response The response to the client.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In c ase of a I/O error encountered
 */
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
{
    // Simply pass pass control to doGet()
    this.doGet(request, response);
    return;
} // end of doPost

/**
 * Processing user's GET request.
 * @param request the client request.
 * @param response the response to client.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOExc eption In case of a I/O error encountered
 */
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
{
    // Initialization
    this.request = request;
    this.response = response;

    // get the session object from the previous servlet
    HttpSession session = request.getSession(true);

```

```

        if(session.isNew() ) System.out.println("^^^% % % % % % % % % % the session is
neww " );

        user = new UserBean() ;

        // get now the UserBean object from the session
        user = (UserBean) session.getAttribute("user");

        response.setContentType("text/html");

        // Get the login parameters from the request
        serverName = request.getParameter("serverID");
        serverPass = request.getParameter("password");
        filename = request.getParameter("filename");
        server = request.getParameter("server") ;

        // validate the server according to the username/passw
        validateServer( server, serverPass ) ;

    } // end of doGet()

    /**
     * Authenticates a server with the given username & password
     * @param name      The server's username.
     * @param pass      The server's password.
     * @return void.
     * @exception ServletException In case of a servlet error encountered
     * @exception IOException In case of a servlet error encountered
     */
    private void validateServer (String name, String pass)
    {

        String permission = "approved" ;           // valid server
        String fileUrl = "nothing" ;                // as a flag in the next

        Vector foundFiles = new Vector(20) ;
        avaFiles = new Vector();

        try
        {
            // the validation is done through the if/else condition
            // It can also be done with a query to the database
            if ( name.equals("http://localhost:8080/metadata/servlet/"
))      avaFiles = findFiles("CIA");
            else if ( name.equals("http://131.120.8.193:8080/metadata/servlet/") )
avaFiles = findFiles("NPS");
            else avaFiles = findFiles("CIA");        //permission="denied" ;

            response.sendRedirect(name + "dbsection.FindUrl?permission="
+ permission +
"&fileUrl=" + fileUrl

```

```

+ filename

+ "&filename="

+ "&server=" + server );

    }
    catch(Exception e)
    {

    }

} // validateServer

/**
 * This method analyses the filename to extract
 * the date and the time, the file saved in the system
 * which actually represents the version of the file.
 * @param fullName The complete name of the file
 * @return An array of string containing the date and the time
 */
private String[] findVersion(String fullName )
{
    char letter ;
    int x =0 ;
    String[] date = { "", "", "", "", "", "", "" } ;

    for(int i=0; i<fullName.length() ; i++)
    {
        letter = fullName.charAt(i) ;

        if (letter != '~')
        {
            date[x] = date[x] + fullName.substring(i, i+1) ;
        }
        else if (letter == '~') x++ ;
    }

    return date ;
}

/**
 * Finds all the files that are releasable
 * to that specific server according to the
 * security authorizations contained in the database
 * @param serverCode The code of this server that recognized by the system
 * @return A Vector containing the releasable files
 * @exception Exception If an error found during the process
 */
private Vector findFiles(      String serverCode)
{

```

```

        openConnection() ;

        // retrieve the available files according to the below criteria
        ResultSet rs4=dbBean.query(
            "SELECT * " +
            "FROM File " +
            "WHERE " + serverCode + " = " + "Yes" + "" );

        try
        {
            while(rs4.next())
            {
                rs4.getString("File");
                avaFiles.add(rs4.getString("UserServiceClass") +
                    rs4.getString("File"));
            }
        }
        catch(Exception e)
        {
            avaFiles = null ;
        }

        return avaFiles ;

    } // findFiles

/**
 * Opens a connection with the database to retrieve
 * the available files of this user
 *
 * @return void
 */
private void openConnection()
{
    dbBean =
(DBConnectionBean)getContext().getAttribute("connectionBean");
    if(dbBean == null)
    {
        dbBean = new DBConnectionBean(); // for the connection module
        dbBean.setDriver("sun.jdbc.odbc.JdbcOdbcDriver");
        dbBean.setUrl("jdbc:odbc:UserDB");
        dbBean.setUser("panos");
        dbBean.setPassword("greek");

        // Load the driver
        if(!dbBean.isLoaded())
        {
            System.out.println("##### driver error ##### = " );
        }
    }
}

```

```
    }  
  
    this.dbBean = dbBean;  
  
    getServletContext().setAttribute("connectionBean", dbBean);  
}  
  
} // end of ServerLogin class
```

```

/*

*****

* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
* Date               :      December 2002

*****

*/

package dbsection;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

/**
 * The UpdateDb class gets an already parsed file,
 * updates the database and saves the file
 * in the "secure" server area.
 */

public class UpdateDb extends HttpServlet
{
    // Data Members
    private UserBean    newuser;
    private      PrintWriter out;
    private HttpServletRequest req;
    private HttpServletResponse res;

    private int    fieldId;
    private String desiredClass;
    private String tempFile;
    private String filename;
    private String finalFilename;
    private      DBConnectionBean    dbBean;

    /**
     * Processing user's GET request by simply passing
     * the control to the doPost.
     * @param request The client request.
     * @param response The response to client.
     * @return void.
     * @exception ServletException In case of a servlet error encountered
     * @exception IOException In case of a I/O error encountered
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)

```

```

throws ServletException, IOException
{
    // Simply pass pass control to doPost()
    this.doPost(request, response);
    return;
} // end of doGet

/**
 * Processing user's POST request.
 * @param request The client request.
 * @param response The response to client.
 * @return void.
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a I/O error encountered
 */
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws IOException
{

    // Initialization
    this.req = req;
    this.res = res;

    // get the session object from the previous servlet
    HttpSession session = req.getSession(true);

    // get now the UserBean object from the session
    newuser = (UserBean) session.getAttribute("user");

    // Get the user selection from the request
    desiredClass = req.getParameter("desiredClass");
    filename      = req.getParameter("filename");

    // Sets the content type to html text
    res.setContentType("text/html");

    // Gets the PrintWriter object for sending HTML commands
    out = res.getWriter();

    //Generates the title
    out.println("<html><head><title>");
    out.println("Update DataBase" );
    out.println("</title></head>");

    //Generates the body
    out.println("<body bgcolor=\"\#fafca3\">");

    printData();           // Print the Personal and the security data
    out.println(" ..... " + "<br>\n");

```

```

        // save the file to the secure server area
        try
        {
            out.println(" Saving the file to the secure server area " +
" <br>\n");
            copyFile() ;
            out.println(" .....Saving the file was successful ...." + " <br>\n");
        }
        catch(IOException e)
        {
            out.println(" ***** Error when copying the file
*****" + " <br>\n");
            out.println(" Please trying again later or call a system afdmin " +
" <br>\n");
            exitLinks() ;
        }

        // update the database
        out.println(" ..... " + " <br>\n");
        out.println(" Updating Database ..... " + " <br>\n");

        openConnection() ;
        boolean successUpdate = updateDatabase() ;

        if(successUpdate)
        {
            out.println(" .....updating database was successful" + " <br>\n");

            //update the available files for that user
            Vector oldAvaFiles = newuser.getAvaFiles() ;
            oldAvaFiles.add(finalFilename) ;
            newuser.setAvaFiles(oldAvaFiles) ;

        }
        else
        {
            out.println(" ***** Error while updating database
*****" + " <br>\n");
            out.println(" Please trying again later or call a system afdmin " +
" <br>\n");
            exitLinks() ;
        }

        exitLinks() ;

        out.println("</body></html>");
    } // doPost ends here

    /**

```

```

* Updates the database with the new file's
*   data and metadata
* @return A boolean variable is returned to verify
*   the success of the procedure
*/
private boolean updateDatabase()
{
    boolean success = false ;

    // Get a new primary key
    dbBean.setPrimaryKeyQuery("SELECT ID FROM File ORDER BY ID");
    int nextKey = dbBean.getPrimaryKey() ;

    if(nextKey != -1)
    {
        fieldId = dbBean.update(
            "INSERT INTO File " +
            "VALUES(" + nextKey + " ,'" + newuser.getServiceName() + "','" +
            """" + newuser.getServicePos() + """" + "','" +
            """" + desiredClass + """" + "','" +
            """" + newuser.getCountryId() + """" + "','" +
            """" + "Yes" + """" + "','" +
            """" + "Yes" + """" + "','" +
            """" + "Yes" + """" + "','" +
            """" + finalFilename + """" + "');"

        success = true ;
    }
    else
    {
        System.out.println(" ERROR getting the primary key " ) ;
    }

    return success ;

} // end of updateDatabase

/**
* Copies the file from the temporary server area to
*   the simulated permanent secure area
* @return void.
* @exception IOException In case of a I/O error encountered
*/
private void copyFile() throws IOException
{
    // create the final filename
    createFinalFilename() ;

    // the URI of the temporary file to read
    String tempFile = System.getProperty("user.dir")
        + "/webapps/metadata/xml/temporary/"

```

```

+ newuser.getUserID() + "temp -1.xml"
;

// the URI to write the file
String finalFile = createUserDirectory()
+ System.getProperty("file.separator")
+ finalFilename ;

File inputFile = new File(tempFile);
File outputFile = new File(finalFile);

FileReader in = new FileReader(inputFile);
FileWriter out = new FileWriter(outputFile);
int c;

while ((c = in.read()) != -1)
out.write(c);

in.close();
out.close();
}

/**
 * Creates the final filename that the file will get in
 * order to be saved, by adding the date and the time
 * @return void.
 */
private void createFinalFilename()
{
    Calendar cal = new GregorianCalendar();
    int year = cal.get(Calendar.YEAR);
    int month = cal.get(Calendar.MONTH) + 1 ; // months start from 0
    int day = cal.get(Calendar.DAY_OF_MONTH);
    int hour = cal.get(Calendar.HOUR_OF_DAY);
    int minute = cal.get(Calendar.MINUTE);
    int second = cal.get(Calendar.SECOND);

    finalFilename = year + "~" + month + "~" + day + "~" +
hour + "~" + minute + "~" + second + "~" +
filename ;

}

/**
 * Creates a new directory according to user's classification
 * in case there is not one yet
 * @return The newly created directory
 * @exception SecurityException In the case of the creation is not
 * allowed by the security manager
 */
private String createUserDirectory() throws SecurityException

```

```

        {
            boolean userDirCreated ;

            File userDir = new File(System.getProperty("user.dir") +
                                   "webapps/metadata/xml/" +
desiredClass ) ;

            // if it is NOT already exists then create it
            if(!userDir.isDirectory() ) userDirCreated = userDir.mkdir() ;

            return userDir.toString() ;

        }

/**
 * Opens a connection with the database to retrieve
 * the available files of this user
 *
 * @return void
 */
private void openConnection()
{
    dbBean =
(DBConnectionBean)getContext().getAttribute("connectionBean");
    if(dbBean == null)
    {

        dbBean = new DBConnectionBean();    // for the connection module
        dbBean.setDriver("sun.jdbc.odbc.JdbcOdbcDriver");
        dbBean.setUrl("jdbc:odbc:UserDB" );
        dbBean.setUser("paulo");
        dbBean.setPassword("silva");

        // Load the driver
        if(!dbBean.isLoaded())
        {
            System.out.println("##### driver error ##### = " );

        }

    }

    this.dbBean = dbBean;

    getContext().setAttribute("connectionBean", dbBean);
}

/**
 * Prints the user's personal and Security data

```

```

* retrieved from the database that contained
* in the UserBean object
* @return void.
*/
    private void printData()
    {
        out.println("<h3>" + " Personal Data " + "</h3>");
        out.println("Last Name : " + "&nbsp;&nbsp;&nbsp;<strong>" +
            newuser.getLastName() + "</strong><br>\n"
+
            "First Name : " + "&nbsp;&nbsp;&nbsp;<s trong>" +
            newuser.getFirstName() + "</strong><br>\n" );

        out.println("<strong>" + " Security Attributes (presented here only for
demo)" + "</strong><br>\n" );
        out.println(" Service Name  : " + "&nbsp;&nbsp;&nbsp;<strong>"
+ newuser.getServiceName() + "</strong><br>\n" +
        " Service Position:" + "&nbsp;&nbsp;&nbsp;<strong>" +
        newuser.getServicePos() + "</strong><br>\n" +
        " Service Classification:" + "&nbsp;&nbsp;&nbsp;<strong>" +
        newuser.getServiceClass() + "</strong><br>\n" +
        " Country ID:" + "&nbsp;&nbsp;&nbsp;<strong>" +
        newuser.getCountryId() + "</strong><br>\n" );
    }

/**
* Displays the available links to direct the control
* in another servlet, in case the user
* wants to cancel this procedure
* @return void.
*/
    private void exitLinks()
    {
        out.println( "<br> <br> <br>\n");

        out.println("<a href=\"" + res.encodeURL("/metadata/html/login.htm")
            + "\">" + "login again as another user " + "</a>" +
" or " + "<br>\n");

        out.println("<a href=\"" + res.encodeURL("UserOptions")
            + "\">" + " see again your options" + "</a>" +
"<br>\n");
    }

}

// UserSelection class ends here

```

```

/*

*****
* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
* Date               :      December 2002
*****

*/

package dbsection ;

import java.util.* ;
import dbsection.*;

/**
 * This class defines a bean used to maintain user's
 * data during the session.
 * @author Panos
 */
public class UserBean
{

    // Data members
    private int userID;
    private String firstName;
    private String middleInitial;
    private String lastName;
    private String street;
    private String city;
    private String zip;
    private String state;
    private String phone;
    private String loginName;
    private String password;
    private String email;

    private String serviceName;           // CIA, NSA, ARMY
    private String servicePos;           // SUser, QUser, Admin, SuperAdmin
    private String serviceClass;         // TS, SE, CL, UN
    private String countryId;
    private Vector avaFiles;

    /**
     * The default constructor initializes a new
     * object by calling the getReset() method
     */
    public UserBean()
    {

```

```

// Initialize properties to a valid state
getReset();
}

/**
 * Resets all fields by creating new empty objects
 * @return An empty String
 */
public String getReset()
{
    userID = -1;
    firstName = new String(" ");
    middleInitial = new String(" ");
    lastName = new String(" ");
    street = new String(" ");
    city = new String(" ");
    zip = new String(" ");
    state = new String(" ");
    phone = new String(" ");
    loginName = new String(" ");
    password = new String(" ");
    email = new String(" ");

    serviceName = new String(" ");
    servicePos = new String(" ");
    serviceClass = new String(" ");
    countryId = new String(" ");

    avaFiles = new Vector();

    return "";
}

/**
 * Gets user's ID
 * @return the user's ID
 */
public int getUserID()
{
    return userID;
}

/**
 * Sets user ID with a new value
 * @param newUserID the new userID
 * @return void
 */
public void setUserID(int newUserID)
{
    userID = newUserID;
}

```

```

/**
 * Gets user's first name
 * @return the user's first name
 */
public String getFirstName()
{
    return firstName;
}

/**
 * Sets user's first name with a new value
 * @param newFirstName The new first name
 * @return void
 */
public void setFirstName(String newFirstName)
{
    firstName = newFirstName;
}

/**
 * Gets user's middle initial
 * @return the user's middle initial
 */
public String getMiddleInitial()
{
    return middleInitial;
}

/**
 * Sets user's middle initial with a new value
 * @param newFirstName The new middle initial
 * @return void
 */
public void setMiddleInitial(String newMiddleInitial)
{
    middleInitial = newMiddleInitial;
}

/**
 * Gets user's last name
 * @return the user's last name
 */
public String getLastName()
{
    return lastName;
}

/**
 * Sets user's last name with a new value
 * @param newFirstName The new last name

```

```

    * @return void
    */
    public void setLastName(String newLastName)
    {
        lastName = newLastName;
    }

    /**
     * Gets user's full name name (first + middle + last)
     * @return the user's full name
     */
    public String getFullName()
    {
        return (getFirstName() + " " + getMiddleInitial() + ". " + getLastName());
    }

    /**
     * Sets user's street name with a new value
     * @param newFirstName The new street name
     * @return void
     */
    public void setStreet(String newStreet)
    {
        street = newStreet;
    }

    /**
     * Sets user's city name with a new value
     * @param newFirstName The new city name
     * @return void
     */
    public void setCity(String newCity)
    {
        city = newCity;;
    }

    /**
     * Sets user's zip code with a new value
     * @param newFirstName The new zip code
     * @return void
     */
    public void setZip(String newZip)
    {
        zip = newZip;
    }

    /**
     * Sets user's state with a new value
     * @param newFirstName The new state
     * @return void
     */

```

```

public void setState(String newState)
{
    state = newState;
}

/**
 * Sets user's phone number with a new value
 * @param newFirstName The new phone number
 * @return void
 */
public void setPhone(String newPhone)
{
    phone = newPhone;
}

/**
 * Sets user's login name with a new value
 * @param newFirstName The new login name
 * @return void
 */
public void setLoginName(String newLoginName)
{
    loginName = newLoginName;
}

/**
 * Sets user's password with a new value
 * @param newFirstName The new password
 * @return void
 */
public void setPassword(String newPassword)
{
    password = newPassword;
}

/**
 * Sets user's email address with a new value
 * @param newFirstName The new email address
 * @return void
 */
public void setEmail(String newEmail)
{
    email = newEmail;
}

/**
 * Sets user's service name with a new value
 * @param newFirstName The new service name
 * @return void
 */
public void setServiceName(String newServiceName)

```

```

{
    serviceName = newServiceName;
}

/**
 * Sets user's service position with a new value
 * @param newFirstName The new service position
 * @return void
 */
public void setServicePos(String newServicePos)
{
    servicePos = newServicePos;
}

/**
 * Sets user's s service classification with a new value
 * @param newFirstName The new service classification
 * @return void
 */
public void setServiceClass(String newServiceClass)
{
    serviceClass = newServiceClass;
}

/**
 * Sets user's country ID with a new value
 * @param newFirstName The new country ID
 * @return void
 */
public void setCountryId(String newCountryId)
{
    countryId = newCountryId;
}

/**
 * Sets user's available files with a new value
 * @param newFirstName The new available files
 * @return void
 */
public void setAvaFiles(Vector newAvaFiles)
{
    avaFiles = newAvaFiles;
}

/**
 * Gets user's street
 * @return the user's street
 */
public String getStreet()
{
    return street;
}

```

```

    }

    /**
     * Gets user's city name
     * @return the user's city name
     */
    public String getCity()
    {
        return city;
    }

    /**
     * Gets user's zip
     * @return the user's zip
     */
    public String getZip()
    {
        return zip;
    }

    /**
     * Gets user's state
     * @return the user's state
     */
    public String getState()
    {
        return state;
    }

    /**
     * Gets user's phone number
     * @return the user's phone number
     */
    public String getPhone()
    {
        return phone;
    }

    /**
     * Gets user's login name
     * @return the user's login name
     */
    public String getLoginName()
    {
        return loginName;
    }

    /**
     * Gets user's password
     * @return the user's password
     */
    public String getPassword()

```

```

{
    return password;
}

/**
 * Gets user's email address
 * @return the user's email address
 */
public String getEmail()
{
    return email;
}

/**
 * Gets user's service name
 * @return the user's service name
 */
public String getServiceName()
{
    return serviceName ;
}

/**
 * Gets user's service position
 * @return the user's service position
 */
public String getServicePos()
{
    return servicePos ;
}

/**
 * Gets user's service classification
 * @return the user's service classification
 */
public String getServiceClass()
{
    return serviceClass ;
}

/**
 * Gets user's country ID
 * @return the user's country ID
 */
public String getCountryId()
{
    return countryId ;
}

/**
 * Gets user's available files stored in the object

```

```

    * @return the user's available files
    */
    public Vector getAvaFiles()
    {
        return avaFiles;
    }

    /**
     * Gets user's personal data all in a String
     * @return the user's personal data
     */
    public String toString()
    {
        String str = new String("User Personal data is:\n");
        str += "UserID    = \t" + userID + "\n";
        str += "First Name = \t" + firstName + "\n";
        str += "Mid initial= \t" + middleInitial + "\n";
        str += "Last name  = \t" + lastName + "\n";
        str += "Street    = \t" + street + "\n";
        str += "City      = \t" + city + "\n";
        str += "Zip       = \t" + zip + "\n";
        str += "State     = \t" + state + "\n";
        str += "Phone     = \t" + phone + "\n";
        str += "Login name = \t" + loginName + "\n";
        str += "Password  = \t" + password + "\n";
        str += "Email     = \t" + email + "\n";

        return str;
    }

    /**
     * Gets user's security data all in a String
     * @return the user's security data
     */
    public String secString()
    {
        String str = new String("User Security data is:\n");
        str += "UserID      = \t" + userID + "\n";
        str += "Sevice Name  = \t" + serviceName + "\n";
        str += "Service Position = \t" + servicePos + "\n";
        str += "Service Classif. = \t" + serviceClass + "\n";
        str += "Country Id   = \t" + countryId + "\n";

        return str;
    }
}

} // End of UserBean class

```

```

/*

*****
* Master Thesis      :      Metadata Security Label Tags
* Author            :      Major (HAF) Aposporis Panagiotis (Panos)
* Advisor           :      Ted Lewis, Ph.D.
* 2nd Advisor       :      Tim Levin
* Date              :      December 2002
*****

*/

package dbsection;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

/**
 * The class UserOptions creates a page displaying the
 * user's information (personal and security) and
 * the available options for that session. After
 * the user's selection passes the control to the next
 * UserSelection servlet
 * @author Panos
 */
public class UserOptions extends HttpServlet
{

    // Data Members
    private UserBean newuser ;
    private Vector avaFiles ;
    private PrintWriter out ;

    /**
     * Processing user's GET request.
     * @param req the client request.
     * @param res the response to client.
     * @return void.
     * @exception ServletException In case of a servlet error encountered
     * @exception IOException In case of a I/O error encountered
     */
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException
    {
        // get the session object from the previous servlet
        HttpSession session = req.getSession(true);

        // get now the UserBean object from the session
        newuser = (UserBean) session.getAttribute("user");

```

```

        String mess = newuser.getFirstName() ;

        // Sets the content type to html text
        res.setContentType("text/html");

        // Gets the PrintWriter object for sending HTML commands
        out = res.getWriter();

        //Generates the title
        out.println("<html><head><title>");
        out.println("User Options" );
        out.println("</title></head>");

        //Generates the body
        out.println("<body bgcolor=\"\#fafca3\">");

        // Print the Personal and the security data
        printData();

        // Generates the form to select an option
        out.println( "<form action='UserSelection' method='POST' > " );
        printOptions();
        out.println( "</form>" );

        out.println("<a href=\"" + res.encodeURL("/metadata/html/login.htm")
                    + "\">" + "login again as another user " + "</a>" +
"<br>\n");

        out.println("</body></html>");

    } // doGet ends here

    /**
     * Prints the user's personal and Security data
     *      retrieved from the database that contained
     *      in the UserBean object
     * @return void.
     */
    private void printData()
    {
        out.println("<h3>" + " Personal Data " + "</h3>");
        out.println("Last Name : " + "&nbsp;&nbsp;&nbsp;<strong>" +
                    newuser.getLastName() + "</strong><br>\n"
+
                    "First Name : " + "&nbsp;&nbsp;&nbsp;<strong>"      +
                    newuser.getFirstName() + "</strong><br>\n" );
        out.println("<strong>" + " Security Attributes  (presented here only for
demo)" +
                    "</strong><br>\n" );
        out.println(" Service Name  : " + "&nbsp;&nbsp;&nbsp;<s trong>"
+

```

```

newuser.getServiceName() + "</strong><br>\n"
+
" Service Position:" + "&nbsp;&nbsp;&nbsp;<strong>"
+
newuser.getServicePos() + "</strong><br>\n"
+
" Service Classification:" + "&nbsp;&nbsp;&nbsp;<strong>" +
newuser.getServiceClass() + "</strong><br>\n"
+
" Country ID:" + "&nbsp;&nbsp;&nbsp;<strong>"
+
newuser.getCountryId() + "</strong><br>\n"
);
    }

/**
 * Creates a drop down list that presents
 * the user's available options.
 * @return void.
 */
private void printOptions()
{
    // insert a horizontal line
    out.println("<hr/>");

    out.println("Your options are ..." + "<br>\n");
    out.println("    <p><select size='1' name='FileMenu'> ");
    out.println("        <option>Import to Save</option>    ");
    out.println("        <option>Open from Database</option>    ");
    out.println("        <option>Import from Other Server</option>    ");
    out.println("        <option>Logout </option>    ");
    out.println("    </select></p>");
    out.println("    <p><input type='submit' value='Submit' name='B1'>
</p>");

}

} // UserOptions class ends here

```

```

/*

*****

* Master Thesis      :      Metadata Security Label Tags
* Author             :      Major (HAF) Aposporis Panagiotis (Panos)
*   Advisor          :      Ted Lewis, Ph.D.
*   2nd Advisor      :      Tim Levin
*   Date              :      December 2002

*****

*/

package dbsection;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

/**
 * The UserSelection class is responsible to get the
 * user's selection from the main menu and to redirect
 * the control to the respective servlet for
 * further processing
 * @author Panos
 */
public class UserSelection extends HttpServlet
{
    // Data Members
    private HttpServletRequest req;
    private HttpServletResponse res;
    private UserBean           freshuser ;
    private Vector             avaFiles ;
    private PrintWriter        out ;
    private String             nextServlet ;
    private String             FileMenu ;

    /**
     * Processing user's GET request by simply passing
     * the control to the doPost.
     * @param request The client's request.
     * @param response The response to the client.
     * @return void.
     * @exception ServletException In case of a servlet error encountered
     * @exception IOException In case of a I/O error encountered
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        // Simply pass pass control to doPost()

```

```

        this.doPost(request, response);
        return;
    } // end of doGet

/**
 * Processing user's POST request.
 * @param req the client request.
 * @param res the response to client.
 * @return void .
 * @exception ServletException In case of a servlet error encountered
 * @exception IOException In case of a I/O error encountered
 */

    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws IOException
    {
        // Initialization
        this.req = req;
        this.res = res;

        // get the session object from the previous servlet
        HttpSession session = req.getSession(true);

        // get now the UserBean object from the session
        freshuser = (UserBean) session.getAttribute("user");

        // Get the user selection from the request
        FileMenu = req.getParameter("FileMenu");

        // Sets the content type to html text
        res.setContentType("text/html");

        // Gets the PrintWriter object for sending HTML commands
        out = res.getWriter();

        //Generates the title
        out.println("<html><head><title>");
        out.println("User Options !!!!!" );
        out.println("</title></head>");

        //Generates the body
        out.println("<body>");

        nextServlet = "/OpenFile" ;
        if (FileMenu.equals("Import to Save") )
        {
            nextServlet = "/ImportToSave" ;
        }
        else if (FileMenu.equals("Open from Database") )
        {

```

```

        nextServlet = "/OpenFile?selFile=" + "start" ;

    }
    else if (FileMenu.equals("Import from Other Server") )
    {
        nextServlet = "/ImportFile" ;

    }
    else if (FileMenu.equals("Logout") )
    {
        nextServlet = "/Logout" ;

    }

    try
    {
        continueSession() ;
    }
    catch(Exception e)
    {
        out.println("Sorry !!! The Exception " + e + " thrown " + "<br>\n" );
    }

    out.println("</body></html>");
} // doPost ends here
/**
 * Continue the session and pass
 *         the control to the next servlet.
 * @return void.
 * @exception Exception. In case an invalid situation is encountered
 */
private void continueSession() throws Exception
{

    // Get the dispatcher
    RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher(nextServlet);

    if (dispatcher == null)
    {
        // No dispatcher means the given file could not be found
        dispatcher = getServletContext().getRequestDispatcher("/UserOptions");
    }
    else
    {
        // Pass control to a different page
        dispatcher.forward(req, res);
    }
} // end of continueSession()

} // UserSelection class ends here

```

LIST OF REFERENCES

- [1] Abrams M.D., Jajodia S., and Podell H. J., *Information Security: An Integrated Collection of Essays*, IEEE Computer Society Press.
- [2] Brown Associates, Inc., Database Trade-offs for IMBM and Oracle: Availability, Scalability, and Performance,” D.H. Associates, Inc., Sep 2001.
- [3] Deitel, H. M., and Deitel, P. J., *Java How to Program*, 3rd Edition, Prentice Hall Inc., 1999.
- [4] Deitel, H. M., Deitel, P. J., and Nieto, T. R., *Internet and The World Wide Web: How to Program*, 2nd Edition, 2002.
- [5] Grohn, M. J., *A Model of a Protected Data Management System*, I.P. Sharp Associates, Jun 1976.
- [6] Hinke, T. H., and Schaefer M., *Secure Data Management System*, Technical Report, System Development Corp., Nov 1975.
- [7] Ibrahim Z., *Mastering the Internet and HTML*, 1st Edition, Prentice Hall Inc., 2000.
- [8] Intelligence Community, Intelink Management Office, Draft Release 0.8, “*Intelligence Community Metadata Standard for Publications, Data Element Dictionary*,” by the Intelligence Community Metadata Working Group, 6 May 2002.
- [9] Intelligence Community, Intelink Management Office, Working Draft, Configuration Management Guide, by the Intelligence Community Metadata Working Group, 27 Sep 2002.
- [10] Kal Ahmet, and others, *Professional XML Metadata*, 1st Edition, Wrox Press Ltd, 2001.
- [11] XML Complete, 1st Edition, SYBEX Inc, 2002.
- [12] Kelly M., “*Guidelines for Intelink Metadata Version 1.0*,” paper presented at Intelink Conference, Newport, RI, Jul 1997.

- [13] Kurose J. F., and Ross K. W., *Computer Networking*, 1st Edition, Addison Wesley Longman, Inc., 2001.
- [14] Li Gong, “*Inside Java 2 Platform Security: Architecture, API Design, and Implementation*,” Addison Wesley, 1999
- [15] Martin B., “*An Introduction to the Extensible Markup Language (XML)*,” [<http://www.personal.u-net.com/~sgml/xmlintro.htm>]
- [16] Marty Hall, *Core Servlets and Java Server Pages*, 1st Edition, Prentice Hall Inc., 2000.
- [17] Microsoft Corporation, “.NET Framework Developer's Guide, .NET Framework Security,” [<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconnetframeworksecurity.asp>], 2001.
- [18] Microsoft Corporation, Project 42 “*An Overview of Security in the .NET Framework*,” Dr. Demien Watkins, Jan 2002
- [19] Microsoft Corporation, “XML Web Services Security,” [<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/xmlwssec.asp>], Feb 2002
- [20] Parnas, D. L., *On the Criteria to Be Used in Decomposing Systems into Modules*, Comm. ACM, Vol. 15, Dec 1972
- [21] PASS Consulting Group, Version 2.0, “IBM DB2 UDB V7.2 and Oracle 9i, A Technical Comparison,” Bloemen J., and others, Frankfurt, 06/09/2001.
- [22] Raj Mohan, “*XML Based Adaptive IPSec Policy Management in a Trust management Context*,” Master of Science Thesis, Department of Computer Science, Naval Postgraduate School, Dec 2002.
- [20] Schafer, M., *Multilevel Data Management Security*, Air Force Studies Board, Committee on Multilevel Data Management Security, National Academy Press, Washington, D.C., 1983
- [23] Refsnes J. E., “Introduction to XML-XSL,” [http://www.xmlfiles.com/xsl/xsl_intro.asp]

- [24] Science Application International Corporation, *Intelligence Community Extensible Markup Language (XML) Final Report*, by Science Application International Corporation XML Study Team, 3 Nov 1999.
- [25] Science Application International Corporation, *Intelligence Community Extensible Markup Language (XML) Prototype System Design Document*, by Science Application International Corporation Applied Content Technologies Team under the directions of the Office of Advanced Analytical Tools, Central Intelligence Agency, UNCLASSIFIED, Revised 28 Aug 2002.
- [26] Science Application International Corporation, *Intelligence Community Extensible Markup Language (XML) Prototype Overview*, by Science Application International Corporation XML Study Team, 6 Aug 1999.
- [27] Science Application International Corporation, *Intelligence Community Extensible Markup Language (XML) Prototype Demonstration Scripts*, by Science Application International Corporation XML Study Team, 30 Sep 1999.
- [28] Scott Oaks, "Java Security," 2nd Edition, O'Reilly & Associates, Inc., May 2001
- Siedschlag M., "*Intelligence Community XML-DTD for Security Markings*," presented at Intelink Conference, Sep 2000.
- [29] Silberschatz A., Korth H. F., and Sudarshan S., *Database System Concepts*, 4th Edition, McGraw-Hill Companies Inc., 2002
- [30] Srinivas R. N., "Java Security Evolution and Concepts, Part 2,"
[<http://www.javaworld.com/javaworld/jw-07-2000/jw-0728-security.html>] , Jul 2000
- [31] SOAP 1.1 Specifications, [<http://www.w3.org/TR/SOAP/>] W3C Note 08 May 2000.
- [32] Sun Microsystems, Inc, *Java 2 Standard Edition, V1.2.2 API Specification*,
[<http://java.sun.com/products/jdk1.2/docs/api/>], Sun Microsystems, Inc., 1999.
- [33] Sun Microsystems, Inc, *The JavaTM Web Services Tutorial*,
[<http://java.sun.com/webservices/docs/1.0/tutorial/>] Sun Microsystems, Inc., 2002.
- [34] Sun Microsystems, Inc, "*JavaTM Security*," [<http://java.sun.com/security/>] Jul 2002

- [35] Wu C. Thomas, An Introduction to Object-Oriented Programming with Java, 2nd Edition, McGraw-Hill Companies Inc., 2000.
- [36] XML Specification, <http://www.w3.org/TR/2000/REC-xml-20001006>, Aug 2002
- [37] XML Schema Specifications, <http://www.w3.org/TR/xmlschema-0>, Aug 2002
- [38] XML Namespace Recommendation, <http://www.w3.org/TR/REC-xml-names/>
- [39] XSLT Specifications, <http://www.w3.org/TR/xslt>, Aug 2002 RFC2396
- [40] Zukowski J., “Exploring the Security Changes of the 1.4 Release of the Java TM 2 Platform Standard Edition (J2SE TM),” Apr 2002

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Commander, Naval Security Group Command
Naval Security Group Headquarters
Fort Meade, Maryland
4. Deborah M. Cooper
Intelligence Community CIO Office
Washington DC
5. Audrey Marsh
Intelligence Community CIO Office
Washington DC
6. William Dawson
Intelligence Community CIO Office
Washington DC
7. Richard Hale
Defense Information Systems Agency, Suite 400
Falls Church, Virginia
8. Cynthia E. Irvine
Computer Science Department, Code CS/IC
Naval Postgraduate School
Monterey, California
9. Timothy Levin
Computer Science Department, Code CS
Naval Postgraduate School
Monterey, California
10. Ted Lewis
Computer Science Department, Code CS
Naval Postgraduate School
Monterey, California

11. Capt Robert Simeral
Security Officer
Naval Postgraduate School
Monterey, California
12. Air Attaché
Embassy Of Greece,
Washington DC
13. Hellenic Air Force General Staff
Education Branch
Athens – Greece